



National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**



National Library
of Canada

Bibliothèque nationale
du Canada

Ottawa, Canada
K1A 0N4

TC -

0-315-22913-6

CANADIAN THESES ON MICROFICHE SERVICE - SERVICE DES THÈSES CANADIENNES SUR MICROFICHE

PERMISSION TO MICROFILM - AUTORISATION DE MICROFILMER

• Please print or type - Écrire en lettres moulées ou dactylographier

AUTHOR - AUTEUR

Full Name of Author - Nom complet de l'auteur

Martin W. Dubetz

Date of Birth - Date de naissance

Canadian Citizen - Citoyen canadien

☐ Yes / Oui

☒ No / Non

Country of Birth - Lieu de naissance

United States

Permanent Address - Résidence fixe

29617 Walker Drive
Warren, Michigan
48092, USA

THESIS - THÈSE

Title of Thesis - Titre de la thèse

Ray Tracing Algorithms for Computer Graphics

Degree for which thesis was presented
Grade pour lequel cette thèse fut présentée

Doctor of Philosophy

University - Université

University of Alberta

Year this degree conferred
Année d'obtention de ce grade

1985

Name of Supervisor - Nom du directeur de thèse

Dr. John Tartar

AUTHORIZATION - AUTORISATION

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

ATTACH FORM TO THESIS - VEUILLEZ JOINDRE CE FORMULAIRE À LA THÈSE

Signature

Martin W. Dubetz

Date

Oct 11/85

NL 91 (1/84/03)

Canada

The University of Alberta

Ray Tracing Algorithms for Computer Graphics

by



Martin W. Dubetz

A thesis

**submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy**

Department of Computing Science

Edmonton, Alberta

Fall, 1985

The University of Alberta

Release Form

Name of Author: Martin W. Dubetz

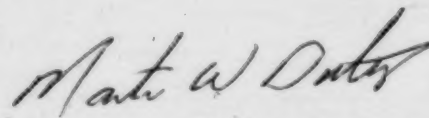
Title of Thesis: *Ray Tracing Algorithms for Computer Graphics*

Degree for Which Thesis was Presented: Doctor of Philosophy

Year This Degree Granted: 1985

Permission is hereby granted to The University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.



Permanent Address: 29617 Walker Drive

Warren, Michigan

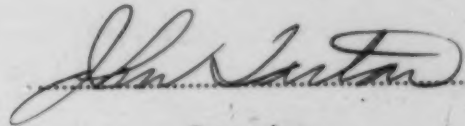
48092, USA

Dated 11 October 1985

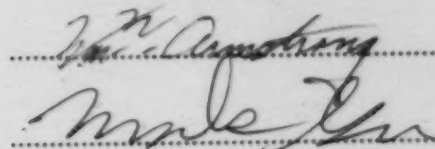
The University of Alberta

Faculty of Graduate Studies and Research

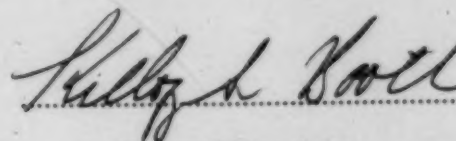
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled *Ray Tracing Algorithms for Computer Graphics* submitted by Martin W. Dubets in partial fulfillment of the requirements for the degree of Doctor of Philosophy.



Supervisor







External Examiner

Date OCT. 10. 1985.

Abstract

Very realistic synthetic images can be produced using a computational device, such as a digital computer. The duplication of real images requires an understanding of the computational nature of image generation and how these processes can be modeled on a such a device. The quality of these images is related to the accuracy of the model used to describe the interaction of light, physical objects, and the eye. This thesis presents an improved lighting model which includes global lighting contributions in the diffuse lighting component of a surface. The efficient production of such images is a major problem in the area of computer graphics. The rate at which these images can be generated is improved by using algorithms, data structures, and architectures that capitalize on the coherence found in the image and parallelism inherent in the image rendering operations. An efficient object/volume data hierarchy for the ray tracing based rendering algorithm is presented in this thesis. An efficient, position independent, smooth shading algorithm and several efficient rendering techniques are also presented.

An illumination model is developed for computer generated images of simple surfaces. This model views a surface as possessing three major lighting components. Two components are reflective terms, one term is weighted by the color of the surface while the other term is independent of surface color. The third term is a transmission component which is weighted by the interior color of the object. The model generally accounts for any distribution of light energy from a surface independently in each term. The model is used to account for diffuse lighting effects from primary and secondary light sources. An efficient and effective implementation of the model is developed, which is based partly on a distributed ray tracing algorithm.

The time required to examine the data primitives in a scene to produce an image

is reduced by using an object/volume data hierarchy. This hierarchy divides the scene by objects until a point is reached where the objects are considered a single part. The part is then divided further using an oct-tree volume hierarchy. The resulting structure is robust and requires minimal modification between frames in an animation sequence. The structure quickly eliminates large portions of the data from consideration by exploiting the spatial ordering of the data and using bounding volumes in the same structure.

A ray tracing algorithm based on the object/volume hierarchy is described. The algorithm is designed to divide the ray tracing task by assigning rays among many processing elements. A simple multi-processor implementation of this algorithm is presented where each processing node contains its own copy of the data and rendering code. This system requires little interprocessor communication and has enough granularity to apply sufficiently large number of processing elements.

Acknowledgements

The author would like to take this opportunity to thank the people who helped make this work possible. As with any research project, numerous people were consulted at all stages of its development. Their time and often insightful criticism of this work sparked new ideas and made old ones clear. In particular I would like to thank the following people.

My first acknowledgements must go to my supervisor John Tartar. During the four years I spent under his supervision I grew both academically and professionally. John's patience and understanding allowed my work to progress further than I could have expected.

Second I would like to thank the members of my examining committee. The personal and technical support that I received from William Armstrong, Kelly Booth, and Mark Green guided the final phases of this research. I would also like to thank Keith Stromsmoe and Tony Marsland for their many helpful suggestions.

Without the resources and technical support provided by The Department of Computing Science at the University of Alberta under the direction Steve Sutphen, this research would not have been possible. Steve's personal interest in my work and his knowledge of the equipment proved invaluable on several occasions.

Finally, I would like to thank Richard Foster for making the math seem so simple, Ted Bentley for, among other contributions, the advice required to format this thesis, and Terry Dubetz for the several readings and encouragement throughout this work.

Table of Contents

Chapter	Page
Chapter 1: Introduction	1
1.1. Definition of Problem	1
1.1.1. Introduction	1
1.1.2. Applications	2
1.1.3. Realistic Image Generation	3
1.2. Research Topic	6
1.2.1. Introduction	6
1.2.2. Image Quality	6
1.2.3. Data Structures	6
1.3. Synopsis of Thesis	7
Chapter 2: Background	8
2.1. Introduction	8
2.2. Lighting Models	10
2.2.1. Introduction	10
2.2.2. Phong's Lighting Model	10
2.2.3. Torrance-Sparrow Model	15
2.2.4. Whitted's Lighting Model	16
2.2.5. Diffuse Lighting	18
2.3. Ray Tracing	20
2.3.1. Introduction	20
2.3.2. Initial Ray Generation and Aliasing	20

2.3.3. Primitives and Intersection Algorithms	24
2.3.4. Secondary Rays and Lighting Trees	29
2.3.5. Smooth Shading	34
2.3.6. Texturing and Bump Mapping	36
2.4. Distributed Ray Tracing	37
2.5. Beam and Cone Tracing	38
2.6. Data Hierarchies	39
2.6.1. Object Hierarchies	40
2.6.1.1. Bounding Volumes	44
2.6.2. Volume Hierarchies	44
2.7. Multiprocessor Ray Tracing	48
2.8. Summary	50
Chapter 3: Lighting Models	51
3.1. Introduction	51
3.2. Improved Lighting Model	52
3.3. Secondary Ray Distribution	58
3.4. Implementation	61
3.4.1. Distributed Ray Tracing	62
3.5. Results	62
3.5.1. Proposed Diffuse Test Image	65
3.6. Summary	70
Chapter 4: Data Hierarchy	76
4.1. Introduction	76

4.2. Object/Volume Hierarchy	77
4.2.1. Construction	77
4.2.1.1. Object Hierarchy	78
4.2.1.2. Volume Hierarchy	80
4.2.2. Traversal	84
4.3. Analysis	91
4.3.1. Introduction	91
4.3.2. Hierarchy Size	91
4.3.3. Rendering	99
4.3.3.1. Construction	100
4.3.3.2. Traversal	104
4.4. Ray Tracing Architectures	109
4.4.1. Introduction	110
4.4.2. Resource Requirements	110
4.4.2.1. Processor Requirements	110
4.4.3. Storage Requirements	113
4.4.3.1. I/O Requirements	115
4.4.4. Multiprocessors	115
4.4.4.1. Parallel Algorithm	116
4.4.4.2. Multiprocessor Architecture	117
4.5. Summary	123
Chapter 5: Conclusions	124
5.1. Introduction	124

5.2. Lighting Model	126
5.3. Object/Volume Hierarchy	126
5.4. Ray Based Architectures	127
5.5. Further Work	127
References	129

List of Tables

Table	Page
3.1 Diffuse Test Data	66
4.1 Volume-Child Relationship	83
4.2 Next Child Index	88
4.3 Test Data Distribution	94
4.4 Data and Node Sizes	94
4.5 Hierarchy Sizes	97
4.6 Construction Timings	101
4.7 Traversal Timings	107
4.8 Processor Timings	111
4.9 Rendering Routines	112
4.10 Timing Summary	113
4.11 Hierarchy Storage Requirements	113
4.12 Active Pages	114

List of Figures

Figure	Page
2.1 Diffuse Lighting	10
2.2 Phong's Lighting Model	13
2.3 Torrance-Sparrow Model	15
2.4 Whitted's Lighting Model	16
2.5 Viewing Parameters	21
2.6 Oversampling Pixels	23
2.7 Ray Polygon Intersection	26
2.8 Shadows	30
2.9 Reflective and Transmitted Rays	31
2.10 Lighting Tree	31
2.11 Smooth Shading	34
2.12 Smooth Shading With Center Normal	35
2.13 Distributed Ray Tracing	38
2.14 Object Hierarchy	41
2.15 Object Hierarchy Traversal	43
2.16 Regular Volume Based Hierarchy	44
2.17 Volume Hierarchy Construction and Traversal	46
2.18 Parallel Ray Tracing Array	48
3.1 Secondary Ray Distribution	60
3.2 Diffuse Lighting on Blocks	64
4.1 Design of the Hierarchy	77

4.2 Object/Volume Hierarchy	81
4.3 Data Subdivision	83
4.4 Traversal Code	88
4.5 Construction Code	90
4.6 Volume Culling	107
4.7 Multiprocessor Architecture	117

List of Photographic Plates

Plate	Page
3.1 Diffuse Lighting Plate 1	71
3.2 Diffuse Lighting Plate 2	72
3.3 Diffuse Lighting Plate 3	73
3.4 Diffuse Lighting Plate 4	74
3.5 Diffuse Lighting Plate 3	75
4.1 Test Images - Plate 1	92
4.2 Test Images - Plate 2	93

Chapter 1

Introduction

1.1. Definition of Problem

1.1.1. Introduction

Computer generated graphics first served to improve the interface between man and machine. Ivan Sutherland's classic paper, published in 1963 and entitled *SKETCHPAD - A Man-Machine Graphical Communication System* [37], defined the fundamentals of computer graphics that have remained valid throughout the last 20 years of development and diversification in the field.

Research in computer graphics proceeded as quickly as equipment to display computer generated images became available. The goal of much of the early research was to improve the quality and effectiveness of information presented to the user. Sutherland's simple line drawings were improved by Roberts [33] with the first algorithm to prevent lines from being displayed which are hidden from view. Next, simple line drawings were augmented with a capability for providing shaded images. The first shaded images were produced using systems developed by MAGI [25] and Appel [2]. During the following years the research efforts in computer graphics divided into interactive computer graphics and realistic image generation, although there is still considerable overlap between the areas. This thesis addresses the latter area, realistic image generation.

The two major problems addressed by this thesis are image quality and rendering efficiency. An improved lighting model is introduced to include global lighting effects in the diffuse lighting component of a surface. An object/volume hierarchy is introduced to reduce the time required to render an image. Using this hierarchy a parallel ray tracing algorithm is specified to efficiently utilize all processors in a large,

multiprocessor system. Other techniques and algorithms are discussed that further reduce image rendering time.

1.1.2. Applications

The first applications to use images that were considered *realistic* was the flight simulation industry. In 1967 General Electric built a system for NASA that was the first one capable of producing shaded images in real time [20], [35]. This system was used to train pilots to fly both aircraft and spacecraft. Cathode ray tubes (CRT) were placed over the windows of a simulated cockpit. Computers were used to generate images on these displays that were as similar as possible to what the pilot would expect to see in a real flight. Similar shading techniques have found application in Computer Aided Design (CAD). The techniques are used to visually present different parts and product designs to designers and engineers for evaluation.

Research in realistic image generation progressed to a point where many objects were so accurately rendered that they appeared almost real. Several new applications for realistic image generation began to appear. The first was in the advertising industry. Graphic artists were able to create images of the products they were selling in new and impressive formats [17]. As the graphics techniques improved, and the cost of the equipment to produce these images decreased, advertisers started producing animation sequences that consisted entirely of computer generated realistic images [27]. Unrestricted by the physical limitations of conventional model building and animation techniques, a new class of logos and short advertising segments appeared in the media. In the last few years the motion picture industry has begun to use realistic image generation to produce movie segments that were impossible or impractical to produce any other way.

Continued improvements in the quality and efficiency of realistic image generation, plus a continued decrease in equipment cost, will open many new application

areas which can use realistic images. The most apparent of these is remote sensing and medical imaging. A system capable of producing realistic images can be interfaced to a conventional sensing system that uses X-rays, ultrasonics, or some other method to detect unseen or inaccessible objects. The sensor data would be converted into a realistic image of the object under examination, greatly improving the amount of information the user can obtain.

Realistic image generation can reduce the need to build physical models and prototypes in the architectural and manufacturing industries when such models are required to obtain a visual preview of a design. Such tools reduce design time and production errors.

1.1.3. Realistic Image Generation

A brief outline of the processes involved in realistic image generation is presented. The intent of this section is to define the problem areas in realistic image generation that this thesis addresses.

The definition and manipulation of the geometric models used to describe a scene were widely researched in the sixties and early seventies. In 1963 Sutherland applied a set of linear geometric operations to manipulate objects in a computer graphics system. The three most common and widely used of these transformations are rotations, scalings, and translations. Roberts [34] applied techniques from projective geometry to obtain a homogeneous coordinate system that allows any combination of linear transformations to be represented by a four-by-four matrix. The three dimensional coordinates of the model are converted to homogeneous coordinates by adding a fourth weighting factor [17]. It has been found that almost all geometric operations applied to objects in computer graphics can be represented by a system of transformation matrices using homogeneous coordinates.

Solid objects were first described using polygons. The polygons used in computer graphics are usually planar surfaces defined by a list of nodes or edges [17]. This object description proved flexible, easy to model, and simple to render. The success of polygons as an effective modeling primitive in computer graphics was due in part to the work done by Gouraud [20] and Bui-Tuong Phong [7] to simulate a curved surface with polygon meshes. Their techniques require substantially fewer polygons to approximate a curved surface than existing methods.

Higher order surfaces, such as quadric and cubic surfaces, are also used to model objects in computer graphics [17]. When they are applicable, a single high-order surface can replace hundreds of polygons. While these surfaces are more compact than polygons, they are difficult to model and render. Many rendering systems convert such surfaces to polygon meshes before rendering.

Stochastic modeling systems, such as particle systems [32] and fractals [8], have been introduced to model natural scenes. Fractal systems have been used to model rough and irregular terrain such as mountains and coastlines. Particle systems have been used to model complex structures such as grass and fire. These models are compact object representations which are expanded using a random factor. The expanded object has physical patterns similar to those found in a natural scene. Fractals are often defined and expanded into polygons while particle systems are defined and expanded into lines. The expanded data then is rendered using the standard algorithms.

Substantial improvements in image quality were obtained by the definition of new lighting models to compute the shading values for the surfaces of an object. Bui-Tuong Phong [7] introduced a lighting model in 1975 that approximated diffuse and specular lighting effects on a surface. Many objects rendered using this model appear almost real. Further improvements were made by Blinn [5] to more accurately model

the specular lighting component of the surface. In 1980 Turner Whitted introduced a lighting model that accounted for reflective and transparent objects and included shadows in a simple and elegant manner [42]. To obtain the required information from a scene, Whitted developed a hidden surface algorithm that traced rays of light in the scene. This technique closely follows the conventional ray optics model used by physicists [22].

Mapping techniques were introduced by Blinn [6] and Newell [3] that added substantial surface detail at little additional computational cost. Texture mapping modifies the color of a surface and bump mapping modifies the surface normals. Both techniques have now become standard parts of realistic rendering systems.

There are still many images that cannot be rendered accurately using these techniques. Some of the lighting effects missing from current images are accurate diffuse lighting, many frequency dependent effects, and light polarization effects. Some progress has been made in a few of these areas, but the results are often not general [26].

Current display technologies present images produced by the computer as an array of color values displayed on a CRT screen or etched onto film. The number of individual locations required in the array to obtain an image of usable resolution is dependent on the physical size of the displayed image. Images recorded on film are often projected onto a large movie screen and require arrays with up to 8,000 by 8,000 (64Meg) values or pixels [17]. CRT displays are much smaller requiring as few as 512 by 512 (256K) pixels for a good resolution image. The rendering system is responsible for setting the color value of each pixel on the display. To perform this operation for a single pixel, each data primitive in the scene must be considered in order to determine those that can influence the final color of the pixel. The data primitives must be sorted by distance from the viewer and those that are visible to the user used to calculate the pixel value. There can be thousands to over a million primitives to consider

per pixel in a high quality image. The key to efficient image rendering is to quickly eliminate large groups of data primitives from further consideration and to efficiently examine the data primitives which must be individually considered.

1.2. Research Topic

1.2.1. Introduction

The two major problems addressed by this thesis are image quality and rendering efficiency. The issue of image quality addresses the problem of accurately modeling diffuse lighting effects. The issue of rendering efficiency defines efficient data structures for organizing the data in the scene.

1.2.2. Image Quality

A new lighting model is presented which includes lighting contributions from secondary light sources in the diffuse lighting component of a surface. This lighting model is more general than existing models and can be expanded to include other lighting effects. Each lighting component of the surface is modeled in a homogeneous manner.

The techniques and algorithms used to effectively implement this model are presented. These techniques make use of a ray tracing algorithm which spatially distributes secondary rays to reduce the number of rays required to produce an image. Several other methods of minimizing the number of rays traced are examined.

1.2.3. Data Structures

A new data structure is presented that organizes the data into an object/volume hierarchy. During rendering the data primitives that do not affect the final value of the pixel are quickly eliminated from consideration. The hierarchy is flexible enough to be used in an animation system with little change in the organization of data with-

5

in the hierarchy from frame to frame.

A parallel ray tracing algorithm is developed that is based on the object/volume hierarchy and can be effectively implemented on a large number of processing elements. The algorithm divides the rendering workload among processing elements by rays. As a scene may contain millions of rays, the algorithm operates efficiently with a large number of processors. It is shown that with sufficiently powerful processors the communication overhead is small and processor utilization over several frames can be very nearly equalized.

1.3. Synopsis of Thesis

The thesis is divided into four additional chapters. Chapter 2 presents the background information related to the main research areas. Chapter 2 presents many of the techniques used to implement the models and algorithms of Chapters 3 and 4. Several of these methods are original and are used to improve the overall performance of the rendering system. As a result of this work, a fully operational ray tracing system has been implemented and used to produce hundreds of images.

Chapters 3 and 4 present the work on the lighting model and the data hierarchy. An attempt is made not to repeat information presented in previous chapters and hence the thesis flows best if the chapters are read in order. Examples, results, and techniques introduced in earlier chapters are freely used in later chapters.

Chapter 5 presents the conclusions that were obtained from this work and possible directions for future research.

Chapter 2

Background

2.1. Introduction

Physicists use two physical models to explain the nature of light and its interaction with physical objects. The first model views light as an infinite number of rays emanating from the light source and traveling into the environment [22]. As the light rays encounter objects they are reflected, refracted, or both. At some point the light rays may enter the eye of an observer where they are visually detected. By modeling the ray as a stream of quantum particles, known as photons, all observable lighting effects can be explained [23]. The second model views light as a wave emanating from the light source and propagating into the environment. As the wave encounters objects it is reflected, refracted, or both, depending on the nature of the object and the laws of classical wave mechanics. The wave properties of light can be derived directly from Maxwell's wave equations for electromagnetism [30]. Due to the wave model's continuous nature and compact mathematical form it is often the preferred theoretical model of light.

Due to the computational complexity of these models, computer generated images have historically been produced using simpler empirical models of light. These simplified models do not directly account for all lighting effects. Recently, new models have been introduced that are partially based on classical ray optics. One such model known as ray tracing [42], includes several lighting effects, such as refractions, in the model in an empirical form.

Some of the most realistic images produced by computer graphics researchers are those rendered using ray tracing algorithms [18]. This realism is directly related to the ability of ray tracing algorithms to obtain global lighting information from the scene for use by complex lighting models. Ray tracing based rendering systems produce

images with the same level of geometric complexity as earlier rendering systems but ray traced images have greater lighting complexity. The lighting characteristics found in complex scenes, such as shadows, reflections and transparencies, are modeled and rendered using a ray tracing algorithm in an elegant, simple, and effective manner [12]. Ray tracing algorithms can accommodate any method of object description where the intersection of a light ray and the object can be determined. The known properties and capabilities of ray tracing algorithms encourage continued research in this area.

The rendering process consists of two major steps. One step is to find the surface that is the closest to the viewer at some particular point on the display. The algorithms used for this step are known as hidden surface algorithms. The other step is to determine the light intensity of this surface. Determining the intensity of a point may require that the intensity of surfaces visible from the point in one or more directions, be determined (global lighting). This requires that the hidden surface and lighting algorithms be applied again. The most effective hidden surface and lighting algorithms for rendering will vary depending on the type of surfaces in a scene. The ray tracing hidden surface algorithm and ray based lighting models effectively render surfaces with global lighting properties.

The different lighting models used to determine the intensity of a surface are examined first. Next the ray tracing algorithm is examined in detail. Techniques for texture mapping, bump mapping, and smooth shading as they apply to ray tracing systems are discussed. Work has been done by other researchers that improves the flexibility and accuracy of the basic ray tracing algorithm. The techniques of distributed ray tracing [12], beam tracing [24], and cone tracing [1] are presented.

The speed of the ray tracing algorithm can be improved by using data structures that allow large portions of the image to be quickly eliminated from consideration during the rendering process. Two such structures that are discussed are object-based

hierarchies and volume-based hierarchies. The efficiency of the ray tracing system can be further improved by the use of special purpose computer architectures. A parallel ray tracing algorithm and the multiprocessor architecture on which it is implemented is presented [40] [10].

2.2. Lighting Models

2.2.1. Introduction

The amount of visual information in an image is determined by the geometric and lighting complexity of the objects in the scene. Lighting models of varying complexity and accuracy have been used in rendering algorithms to capture the lighting effects present in a scene. Simple lighting models can produce accurate images of scenes with low lighting complexity, but scenes with complex lighting characteristics require more complex, and often more costly, lighting models.

This section examines several lighting models in order of increasing complexity. Each model can accurately represent a particular class of scenes. More complex models include a larger class of images and often a large portion of the simpler models. All the models presented in this chapter are intensity based. Color effects are obtained using the relative intensities of the three additive primary colors (red, green, and blue). Under some conditions this technique does not correctly account for frequency based lighting effects.

2.2.2. Phong's Lighting Model

Most surfaces encountered in the real world have a large lighting component that diffusely reflects light. Figure 2.1 shows a perfectly diffuse surface reflecting incident light evenly in all directions on the lighted side of the surface.

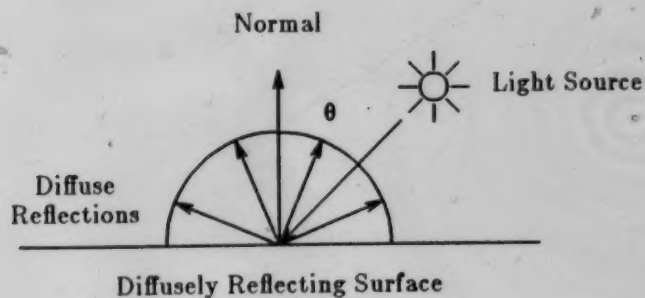


Figure 2.1 Diffuse Lighting

The intensity of the reflection is dependent only on the intensity and relative position of the light source to the surface. Light sources directly above the surface have the greatest reflected intensity. As the light source moves to one side or the other, the reflected intensity decreases. This relationship, known as Lambert's cosine law [17], is shown in Equation 2.1.

$$I_D = I_{LS} K_D \cos(\theta) \quad 2.1$$

where,

I_D = The reflected intensity of the diffuse component.

I_{LS} = The intensity of the light source.

K_D = The diffuse coefficient of the surface.

θ = The angle between the light source and the surface normal.

The diffuse coefficient, K_D , is the percentage of incident light that is reflected from the surface in a diffuse manner. Examples of surfaces with large diffuse coefficients include rocks, paper, cloth, and wood. Diffuse reflections are mainly due to scattering of light in, and on, the surface. If the surface has different reflectances for light of different frequencies the surface will appear colored [27]. In this model the

term K_D is the average reflectance of all frequencies of light. The color of the object is specified as weights of the three additive primary colors. This approach determines a single intensity value for the surface which is weighted by the color of the object and light source as a final step.

Light can be obtained from many sources. Objects that emit light are known as primary light sources. Light is also obtained through reflections from other objects in the scene. These objects are considered to be secondary light sources. The application of Equation 2.1 requires that each light source be modeled as a point to correctly determine the value of θ . Primary light sources are often specified as point light sources or approximated by several point light sources when using this model. Secondary light sources are usually not considered.

Computer images produced using only Equation 2.1 appear harsh. This is partially because the contributions from the secondary light sources are not included. The light from the secondary light sources often is approximated by assuming the secondary lighting contribution is independent of direction and to have an average color of white. To account for light modeled with this method, a simple constant term is added to Equation 2.1. This term accounts for diffusely reflected secondary light and as such is weighted by the color of the surface. Equation 2.2 includes an ambient lighting term for modeling diffuse reflections from secondary light sources.

$$I_D = I_A + I_{LS} K_D \cos(\theta) \quad 2.2$$

where,

I_A = The ambient lighting value.

Surfaces also reflect a component of incident light in nearly a single direction relative to the angle of incidence. This lighting component, known as specular lighting, is unaffected by the color of the surface and retains the color of the incident light.

Examples of objects that demonstrate this lighting effect are the shiny spots on an apple, the gloss from a floor, and the reflection from glass. A perfect mirror reflects this light in a single direction, whereas rougher surfaces reflect light in a range of directions about the major reflective axis. The roughness of the surface and hence the spread of the light about the main reflective direction has been empirically modeled by a cosine function raised to a power [7]. The higher the power, the more perfect the reflection. This model has no theoretical bases, yet produces realistic specular reflections. The amount of specular light reflected from a surface is given by Equation 2.3.

$$I_S = I_{LS} K_S \cos^n(\phi) \quad 2.3$$

where,

I_S = The intensity of the specular light reflected towards the viewpoint.

K_S = The specular coefficient for the surface.

ϕ = The angle between the viewer and the reflective direction.

n = An exponent controlling the spread of specular light.

The term K_S is the percentage of light specularly reflected from the surface. As with Equation 2.1, Equation 2.3 accounts for light from a single point light source only.

The ambient, diffuse, and specular components of a surface are combined into a single equation that is known as Phong's Lighting Model [7]. Figure 2.2 shows the relevant parameters required to determine the diffuse and specular components of a surface for this model.

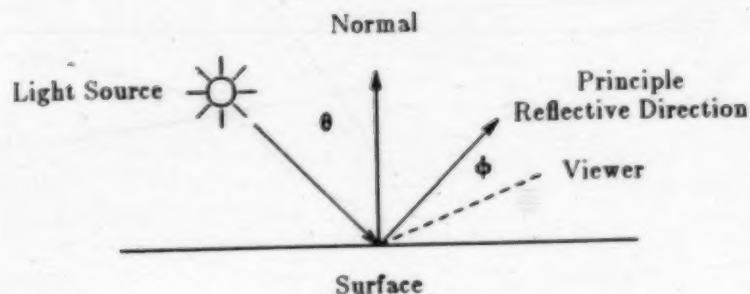


Figure 2.2 Phong's Lighting Model

Equation 2.4 is Phong's Lighting Model, including the ambient, diffuse, and specular components. The terms have been modified to include contributions from multiple point light sources by summing the individual lighting contributions.

$$I = I_A + K_D \sum_{i=1}^{N_L} I_i \cos(\theta_i) + K_S \sum_{i=1}^{N_L} I_i \cos^n(\phi_i) \quad 2.4$$

where,

N_L = The number of light sources.

I_i = The intensity of the i^{th} light source.

θ_i = The angle between the surface normal and the i^{th} light source.

ϕ_i = The angle between the viewer and the i^{th} reflective direction.

Phong's lighting model is used in scanline rendering systems with satisfactory results. Simple improvements to the model, such as varying the intensity of the light source based on the distance the light source is from the surface, and the inclusion of an ambient lighting component that is unweighted by the color of the surface, have been used, but only slightly improve the quality of the final image. Phong's model produces accurate images of a large class of scenes. The only information required to apply this model is the intersecting surface, to obtain the color of the object, and the

location of the point of interest on the surface, to determine the values of the $\cos(\theta)$ and $\cos(\phi)$.

2.2.3. Torrance-Sparrow Model

A theoretically based lighting model for reflective surfaces was proposed by Torrance and Sparrow [39] in 1967 to explain the off-specular reflections from roughened surfaces. This model was adapted to computer graphics by Blinn [5] in 1977.

The surface of the object is modeled as a collection of microscopic facets, each acting as a mirror. The specular component is due to mirrors that reflect light directly towards the viewer. The diffuse component of a surface is due to scattering and multiple reflections between facets. Figure 2.3 shows a surface using this model. Two facets, a and c, are reflecting specular light towards the viewer.

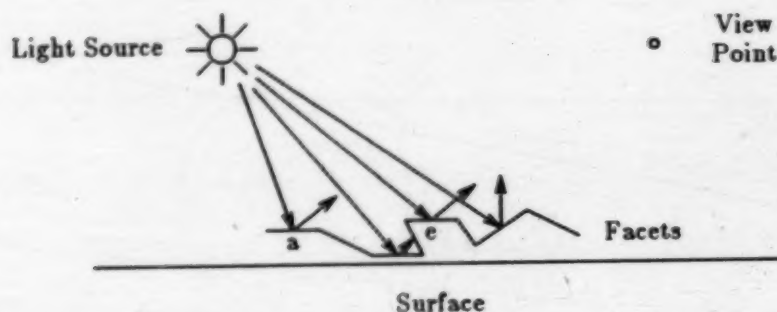


Figure 2.3 Torrance-Sparrow Model

Each facet has a local normal. The percentage of facets with the appropriate normal which will reflect light towards the viewer is modeled with a simple Gaussian distribution that is a function of the angle between the viewer and the normal to the surface. The amount of light reflected and absorbed by a facet is modeled by the Fresnel reflection law [39]. This part of the model accounts for the near perfect reflections from facets when the light impinges at a low angle. Also considered is the amount

that a facet shadows or masks another facet. Blinn [5] defines and presents formulations for each of the parameters of the Torrance-Sparrow model as applied to computer graphics. The reader is directed to Blinn's paper for further details.

The Torrance-Sparrow model is more accurate than the model presented by Phong and is nicely based on a theoretical model of the surface. The most notable improvements occur when the light strikes the surface at shallow angles. The Torrance-Sparrow model as defined by Blinn, only includes the lighting contributions from the primary point light sources in the specular lighting term.

2.2.4. Whitted's Lighting Model

The models presented thus far are used to produce realistic images of a large class of objects. These images contain only point light sources and surfaces with diffuse and specular components. The interaction of light between objects is limited to a simple background value, with two important exceptions. Techniques exist for including shadows [14] and nonrefractive transparent objects [7] in such scenes to obtain greater realism. Further techniques are now presented that improve the degree that the interaction of light between objects is modeled.

Work by Blinn and Newell [3] and Whitted [42] improved the specular term of the lighting model by including the lighting contribution from secondary light sources in the scene. To obtain the required information for his lighting model, Whitted developed a ray tracing algorithm. Whitted's algorithm traces light rays from the eye into the scene creating reflective and transmitted rays when striking a surface. The extra rays are required to determine the global lighting contribution to the intensity of a point on a surface. A complete discussion of ray tracing algorithms is presented in Section 2.3. Figure 2.4 shows the different rays and parameters used in Whitted's lighting model.

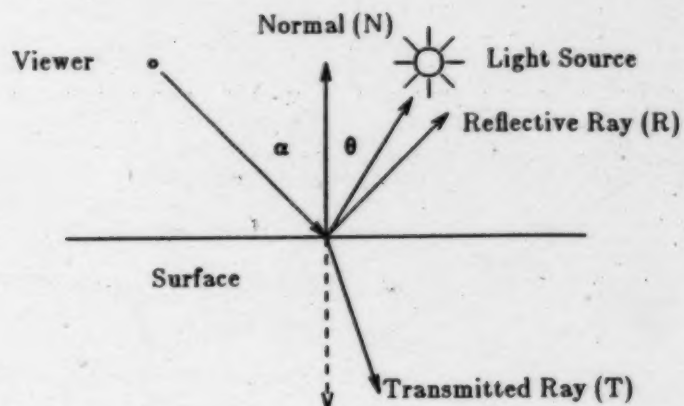


Figure 2.4 Whitted's Lighting Model

The direction of the reflective ray is a function of the angle between the viewer and the normal to the surface (α). The direction of the transmitted ray is a function of the index of refraction of the material on both sides of the surface and the angle α . Whitted's lighting model is shown in Equation 2.5.

$$I = I_A + K_D \sum_{i=1}^{N_L} I_i \cos(\theta_i) + K_R R(\alpha) + K_T T(\alpha) \quad 2.5$$

where,

K_R = Reflective coefficient of the surface.

$R(\alpha)$ = The intensity in the reflected direction from a surface.

K_T = Transmission coefficient of the surface.

$T(\alpha)$ = The intensity in the transparent direction into a surface.

The first two terms of Whitted's lighting model account for the diffuse component of the surface. These diffuse terms are identical to Phong's Lighting Model. The last two terms of Equation 2.5 model surfaces which reflect or transmit light in a single direction. The model accounts for contributions from secondary light sources so

accurately that images of these objects are clearly visible on the reflective surface and objects behind a transparent surface are correctly distorted.

2.2.5. Diffuse Lighting

The lighting models used by Phong, Torrance-Sparrow, and Whitted accounted for the effects of secondary light sources in the diffuse component with a simple constant term. A radiosity method was proposed by Goral, Torrance, Greenburg, and Battaile [19] to more accurately account for the contributions from secondary light sources in the diffuse lighting calculations. These lighting effects include soft shadows and color bleeding between nearby surfaces.

Goral, Torrance, Greenburg, and Battaile define the amount of light which leaves a surface as the sum of the light emitted from the surface and a sum of all the light reflected from other objects in the scene which falls on the surface. The sum of the reflected light is weighted by the reflectance of the surface. The light emitted by the other objects in the scene is, in part, dependent on the light emitted from the surface under consideration. This results in a complex inter-relationship between objects. The intensity of a surface in this model is shown in Equation 2.6.

$$I_j = E_j + K_{D_j} \sum_{i=1}^{N_S} F_{i,j} I_i \quad 2.6$$

where,

I_j = The intensity of surface j .

E_j = The intensity of the light emitted from surface j .

K_{D_j} = Diffuse coefficient of surface j .

N_S = Number of surfaces in the scene.

$F_{i,j}$ = Fraction of light leaving surface i and falling on surface j .

The calculation of the form factor $F_{i,j}$ was improved by Cohen and Greenburg [11] to more effectively model complex environments.

To determine the objects that reflect light towards a particular surface a hemi-cube is constructed around the surface. The objects in the scene are projected onto the hemi-cube using a standard hidden surface algorithm. For each projected object a form factor is calculated to determine the percentage of light radiating from the object that strikes the hemi-cube. If the object is hidden from view there will be no projected image on the hemi-cube and the form factor is zero. These values are used in Equation 2.6 to determine the amount of light radiating from the surface. This process is repeated for each surface.

The result of this procedure is a set of equations that must be solved simultaneously. There is one equation for every surface. Each equation has a term to include contributions from each surface in the scene. If N_S is the number surfaces in the scene then a matrix solution of the equations requires a N_S by N_S matrix. A single lighting value is obtained for each surface by solving this set of equations. If the light intensity is to change across a surface, as is common for smoothly shaded images, the surface must be divided into several smaller pieces, increasing the number of equations. While the generation and solution of these equations is costly, the resulting lighting solution is valid for any viewing position, but must be recalculated if any objects are moved. The model cannot account for reflective, transparent, or specular lighting effects in the scene.

A similar diffuse lighting model was developed by Nishita and Nakamae [28]. Primary light sources are treated separately to allow point light sources to be used. To determine the objects which reflect secondary light to a particular surface a hidden surface algorithm similar to a shadow algorithm is applied. A set of equations, identical to Goral, Torrance, Greenburg, and Battaile is obtained and solved.

2.3. Ray Tracing

2.3.1. Introduction

The ray tracing algorithm plays an important role in realistic image rendering. The first step of the ray tracing algorithm is the generation of initial rays. The number and direction of the initial rays is dependent on the location of the viewer, the direction of view, the field of view, and the geometric complexity of the scene. Each ray is effectively compared with each primitive to find the closest primitive to the origin of the ray that intersects the ray. Secondary rays are generated to obtain shadow information and global lighting information for reflective and refractive surfaces. Conventional lighting models, smooth shading, and mapping algorithms can be applied to the information collected by the ray tracing algorithm. More powerful lighting models that use distributed ray tracing techniques [12] can account for effects such as penumbra shadows and motion blur.

2.3.2. Initial Ray Generation and Aliasing

A set of initial rays are generated with their origin at the view point and directed to pass through the imaginary display polygon. Rays are described by several methods. One method used is to represent a ray in its parametric form. This form is particularly convenient because the value of the parametric variable is linearly related to the distance from the origin of the ray. Equation 2.7 is the parametric form of a ray originating at point P_0 and passing through point P_1 .

$$R(t) = (1 - t) P_0 + t P_1 \quad 2.7$$

where,

$R(t)$ = A vector representing a point at t along the ray.

t = The parametric variable.

P_0, P_1 = Two vectors representing the origin and a point along the ray.

The vectors R , P_0 , and P_1 have three components in a three dimensional coordinate system. The position along the ray is determined by the value of t . The value of $R(t)$ is P_0 when t is zero and P_1 when t is one. Equation 2.7 is valid for positive values of t . Negative values indicate positions behind the origin of the ray.

The first step in the ray tracing process is to define the view. The view is specified with five parameters. The first parameter is the location of the view-point in three dimensional space. The second and third parameters are the direction of the view and the direction considered up. These parameters are specified with direction cosines. The last two parameters are the field of view along the x axis and the aspect ratio of the display polygon. These two parameters define the shape and size of the viewing pyramid. The field of view and the aspect ratio alternatively can be specified as two viewing angles one along the x axis and one along the y axis. Figure 2.5 shows one definition of the variables required for the viewing parameters.

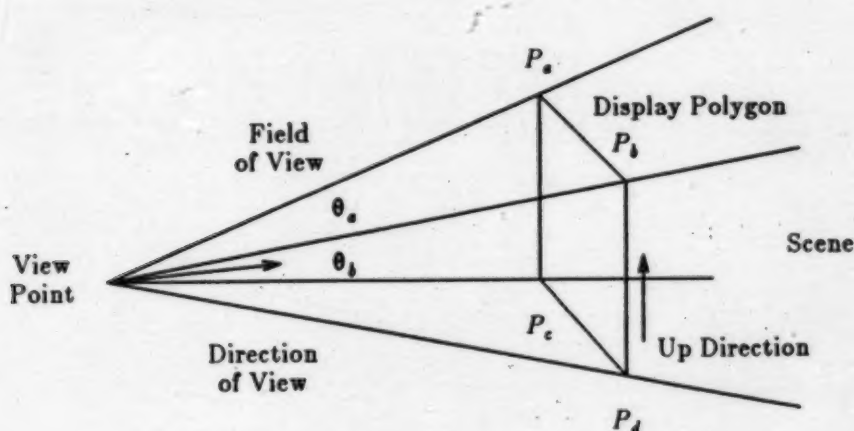


Figure 2.5 Viewing Parameters

The display polygon shown in Figure 2.5 is obtained by using the five viewing parameters and an arbitrary distance from the viewpoint. For convenience the display

polygon is placed such that the distance from the viewpoint to the center of the display polygon is one. It is useful to represent this polygon parametrically. Once the four corner points have been determined a parametric equation of two variables can be defined as shown in Equation 2.8.

$$P(u,v) = (1-u)(1-v)P_a + (1-u)vP_b + u(1-v)P_c + uvP_d \quad 2.8$$

where,

u, v = The parametric variables.

P_a, P_b, P_c, P_d = The four corner points of the display polygon.

The values of u and v range from zero to one inclusive. In many instances the display polygon must be divided into smaller pieces. This is done by simply specifying ranges for parametric values u and v .

The ray tracing algorithm will project an image onto the display polygon. The display polygon is mapped directly to the display device or raster file for presentation or storage. The aspect ratio of the display polygon is identical to the intended mapping area on the display. Frequently the display polygon is mapped to the entire display. The display and raster file are configured as an array of n by m square pixels of equal area. Rays are traced from the viewpoint through the display polygon to obtain the intensity values for each of these pixels.

Ray tracing is a sampling technique and hence errors will result if insufficient samples are taken. According to sampling theory [13] the intensity of each pixel should be set to a weighted average of all the sampled scene intensity in and around the pixel. The sampling pattern is designed as to minimize the visual errors observed on the display.

Sampling errors occur when the image contains spatial frequencies that are more than half the sampling rate. These errors are apparent in the final image as noise or

aliasing effects. Aliasing effects occur where the image contains high frequency components and is sampled in a regular pattern. Aliasing errors are also known as the staircase or jagged line effects. Physically, aliasing effects occur when only part of the pixel is covered by the object. Depending on where the sample is taken, the object may, or may not, be included in the final intensity calculation for the pixel. The amount of error in a pixel's value can be reduced by oversampling the pixel with more rays. An effective pattern for sampling an image is to sample through the corners of the pixels as mapped onto the display polygon. Several oversampling patterns at varying rates are shown in Figure 2.6. A weighting function is applied to the sample points to determine the final intensity of the pixel.

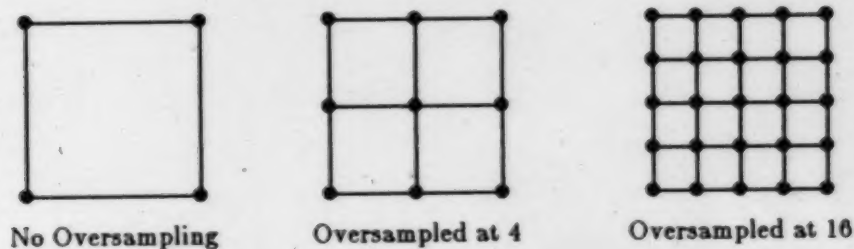


Figure 2.6 Oversampling Pixels

The sampling pattern also can be stochastically determined using a Poisson distribution or by jittering a regular sampling pattern. The sampling error that results from undersampling will appear as noise effects. These errors appear as small spots in the final image [15]. Many viewers find this type of sampling error less annoying than aliasing effects.

Oversampling is necessary only in the areas of the scene where there are high frequency components, such as along the edges of objects. The oversampling rate can be controlled dynamically by creating oversampling rays only when a large difference is

found between adjacent sample values. This requires an initial pattern of rays to be traced to determine where oversampling is required. Often a single ray is traced per pixel to obtain this initial sample.

Another form of sampling error occurs when small objects appear and disappear from frame to frame in an animation sequence as they are struck or missed by rays. This effect can be eliminated by tracing one additional ray to the center of each object and using a dynamic sampling procedure to reduce sampling errors.

2.3.3. Primitives and Intersection Algorithms

The object closest to the origin of the ray that intersects the ray, is determined by comparing the ray to each primitive in the scene. First, it must be determined if the ray intersects the primitive. This operation can be performed at varying levels and degrees of certainty. The exact point of intersection with the primitive must be known for each primitive which intersects the ray. This point is required to determine which primitive is closest and is used to perform shading and mapping calculations.

There is a series of tests that can be used to eliminate large groups of primitives from consideration. These are fail-only tests and do not guarantee that the ray intersects the remaining primitives. These tests often rely on the data being organized in some particular fashion and hence require some degree of preprocessing. The details of these tests are examined shortly. The algorithms used to determine the exact point of intersection for individual data primitives are considered first.

The most common and versatile primitive is the planar polygon. Non-planar polygons can be used but often present serious problems when viewed from an angle where they appear folded. Meshes of polygons are used to approximate objects to any desired degree of accuracy. For man-made objects, simple polygon meshes often describe the object exactly. There are two main methods used to describe a polygon. The first method uses a list of points and edges to delimit the polygon. The second

method is a parametric representation identical to that used for the display polygon.

Polygons which are described by their edges can take on several forms. One efficient method is to store all the unique nodes or corners of the polygons in a linear list. The polygon is defined as a list of nodes which are connected by edges which outline the polygon. The last point of the list is assumed to be connected to the first to close the polygon. To intersect a ray with a polygon described in this manner the ray is first intersected with the plane of the polygon. The plane of the polygon is defined as the plane where all the data points reside and can be determined from the list of nodes in the polygon using Equation 2.9 [4].

$$A = \sum_{i=1}^N (y_i - y_{i+1})(z_i + z_{i+1}) \quad 2.9a$$

$$B = \sum_{i=1}^N (z_i - z_{i+1})(x_i + x_{i+1}) \quad 2.9b$$

$$C = \sum_{i=1}^N (x_i - x_{i+1})(y_i + y_{i+1}) \quad 2.9c$$

$$D = -(A x_1 + B y_1 + C z_1) \quad 2.9d$$

where,

A, B, C, D = The coefficients of the plane equation.

N = The number of nodes in the polygon.

x_i = The x coordinate of point i .

y_i = The y coordinate of point i .

z_i = The z coordinate of point i .

When the value of $(i + 1)$ in Equation 2.9 is larger than N it assumes the value of 1. The coefficients of the plane equation are often normalized so that the terms A, B , and C represent a unit normal to the surface. The plane equation is determined only once for each polygon and is stored with the description of the polygon for easy access.

The description of the ray given by Equation 2.7 is used to intersect the plane. The result of this calculation is the parametric value of t along the ray where the polygon and the ray intersect. This value is found using Equation 2.10.

$$t = - \frac{(A P_{0x} + B P_{0y} + C P_{0z} + D)}{(A (P_{1x} - P_{0x}) + B (P_{1y} - P_{0y}) + C (P_{1z} - P_{0z}))} \quad 2.10$$

If the denominator of Equation 2.10 is zero the ray is parallel to the plane. If the value of t is negative the intersection occurs behind the origin of the ray and is discarded. If t is positive and smaller than the value of t for the closest primitive found so far a second test must be done to determine if the ray intersects the plane inside or outside the perimeter of the polygon.

To determine if the ray intersects the plane inside the polygon it is both convenient and efficient to project the polygon and the intersection point onto one plane of the coordinate axis. The three planes of projection are the $x = 0$, $y = 0$, $z = 0$ planes. This projection involves simply dropping the coordinate component that is equal to zero on this plane. The projection plane selected should have the largest projected area of the polygon to obtain the most accurate calculation and avoid singularities. Such a plane can be found by determining which component of the normal to the polygon is the largest. The normal for the polygon is the vector of the first three terms of the plane equation. If A is larger than B and C for a given polygon the projection plane would be $x = 0$. Figure 2.7a shows a ray intersecting a polygon and its projection onto one of the coordinate planes.

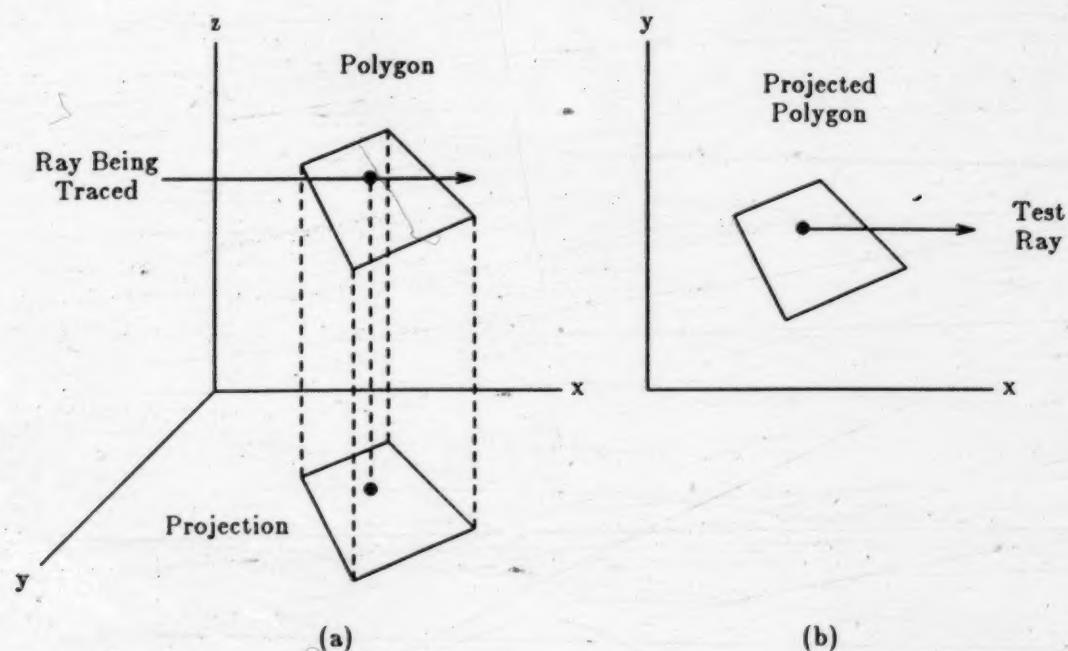


Figure 2.7 Ray Polygon Intersection

To determine if the intersection is inside or outside the polygon, an arbitrary ray is drawn from the intersection point on the projection plane as demonstrated in Figure 2.7b. This ray should be chosen parallel to one of the coordinate axis for efficiency. Each edge is considered to be a ray with P_0 at one end and P_1 at the other. The ray is then compared to each edge using Equations 2.11a and 2.11b. These equations assume the projection was on the $z = 0$ plane.

$$t_0 = (P_{0x} - I_x) + t(P_{1x} - P_{0x}) \quad 2.11a$$

$$t_1 = (I_y - P_{0y}) + t(P_{1y} - P_{0y}) \quad 2.11b$$

where,

t_0 = The distance along the ray where it intersects the edge.

t_1 = The distance along the edge when it intersects the ray.

I_x, I_y = The intersection point on the plane.

P_0, P_1 = The end points of the edge under test.

If t_0 is less than zero then the ray intersects the edge behind the origin of the ray and is ignored. If t_1 is less than zero or greater than one then the ray intersects the edge outside its length and no intersection is counted. The ray is compared to each edge and the number of intersections counted. If there are an odd number of intersections the point of intersection is inside the polygon. This algorithm requires the polygon be planar and non-folded. There are several similar methods used to determine if the intersection point is inside the perimeter of the polygon. One method treats each edge as a plane that is perpendicular to the polygon. If the intersection point is on the same side of each plane it is inside the polygon. Another method translates the polygon and intersection point such that the intersection point is on the origin. The x axis is used to intersect each edge. An odd number of intersections indicates the intersection point is inside the polygon.

The second method of polygon intersection requires all polygons to be four sided and represented in parametric form. Only polygons of four or fewer sides can be generally represented parametrically. The four nodes are used to parametrically define the polygon using Equation 2.8. The intersection of a ray with a parametrically defined polygon begins by intersecting the ray with the plane of the polygon to obtain the intersection point. The equations to determine the parametric values of u and v from the parametric equation and the intersection point were derived by Ullner [40] as part of the design of his parallel ray tracing machine. The equations require numerous operations including a square root. If the values of u and v are both between zero and one, the intersection point is inside the polygon. The parametric values of the polygon obtained by this algorithm are often used later during the mapping operations.

Higher order surfaces, such as quadrics and cubic surfaces, are also used to model objects. A single such surface can often replace hundreds of polygons. Quadric

surfaces are second order surfaces of three variables. Examples of quadric surfaces are spheres and cylinders. Cubic surfaces are third order surfaces of three variables. Cubic surfaces are an extension of splines and can be defined so that they smoothly join into meshes [17]. These surfaces can be described homogeneously and delimited by planes or defined as patches and represented parametrically. The homogeneous form is simpler to intersect with a ray but it is difficult to join two such surfaces smoothly. Parametrically defined surfaces are more difficult to intersect and, in the case of cubics, have no closed form solution, but are easy to model. In practice quadric surfaces are represented homogeneously whereas cubics are defined parametrically. For a further definition of these surfaces the reader is directed to the text by Foley and VanDam on interactive computer graphics [17].

Any primitive can be used to describe an object as long as it can be intersected with a ray such that the relative distance from the origin of the ray to the point of intersection can be determined.

2.3.4. Secondary Rays and Lighting Trees

Secondary rays are generated when a ray intersects an object. These rays originate at the point of intersection on the surface and are directed into the scene to obtain the global lighting information. The number, type, and direction of the secondary rays is dependent on the lighting model used and the characteristics of the surface. The evaluation of the surface intensity using the information collected by the rays is viewed as traversing a lighting tree that contains the points of ray-surface intersections at the nodes and represents the rays as links between nodes. The types of secondary rays and their functions are presented first, followed by a discussion on evaluation of the lighting tree.

The first secondary ray generated is often the shadow ray. Shadows greatly improve the realism of the scene being rendered and are used extensively in many

rendering systems. Shadows are caused by objects that block a particular light source from making a contribution to the intensity of a point on the surface of an object. Let the variable S_i assume the value zero if there is an object between the intersection point and the light source i and assume the value one otherwise. Whitted's lighting model, given by Equation 2.5 can be rewritten, as shown in Equation 2.12, to account for shadows.

$$I = I_s + K_D \sum_{i=1}^{N_{LS}} S_i I_i \cos(\theta_i) + K_R R(\alpha) + K_T T(\alpha) \quad 2.12$$

The value of S_i is determined by generating a secondary ray from the point of intersection to each light source as demonstrated in Figure 2.8.

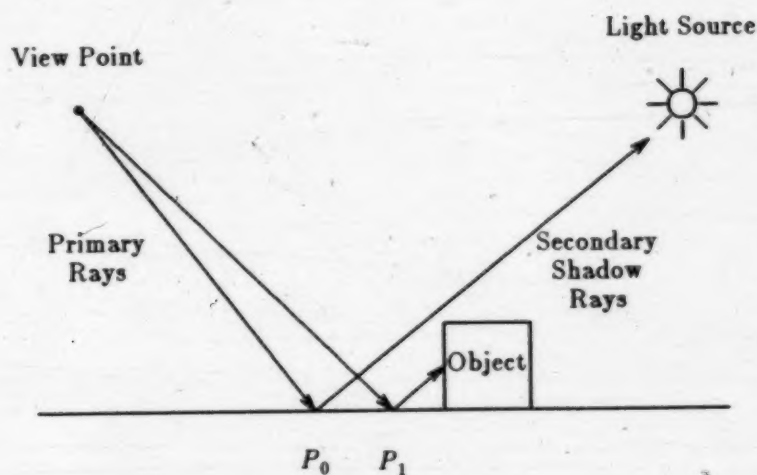


Figure 2.8 Shadows

The shadow ray is traced until a primitive is found that the ray intersects, the ray tracing process is then stopped and S_i for this light source is set to zero. If no primitive is found that intersects the ray then S_i is set to one. The point P_0 in Figure 2.8 is not in a shadow whereas point P_1 is in the shadow of the box. From this procedure it can be seen that scenes with many shadows may be rendered more quickly

than a comparable scene with few shadows.

If the surface is reflective or transparent, secondary rays are generated to obtain the lighting information for these components of the lighting model. Figure 2.9 shows several surfaces that require both reflective and transparent rays to be traced. The sphere is partially reflective and partially transparent whereas the two boxes are only reflective.

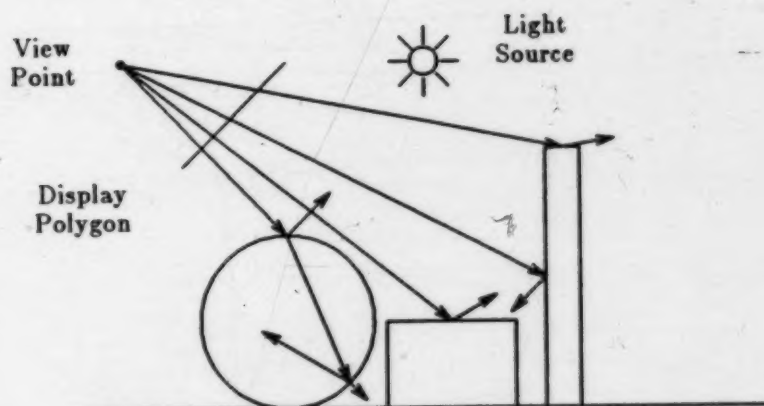


Figure 2.9 Reflective and Transmitted Rays

The direction of the reflective rays is such that the angle between the normal and the incoming ray, and the normal and the reflected ray, are of the same magnitude but of different signs. The transmitted ray is deflected depending on the index of refraction of the material on both sides of the surface and the angle between the normal and the incoming ray [42].

The reflective and transparent rays may strike other surfaces causing more rays to be generated. Figure 2.10 shows a partial path of a ray and the virtual lighting tree which would be created. The creation and evaluation of this tree is considered next.

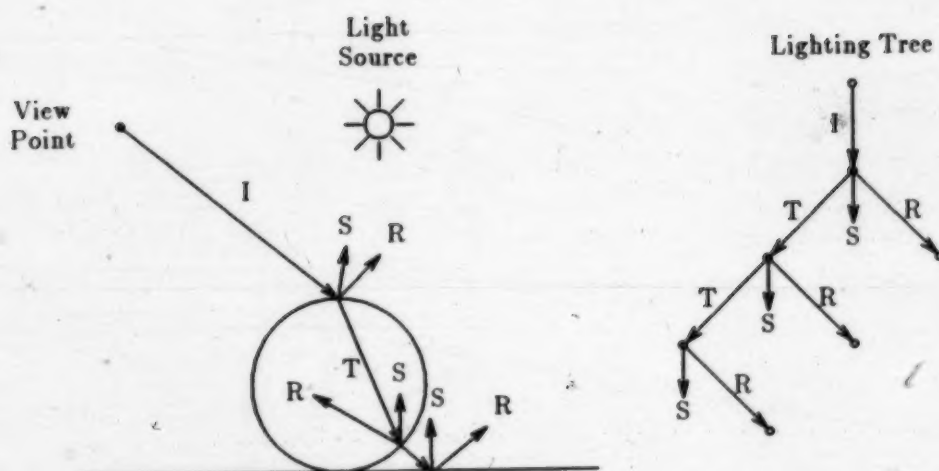


Figure 2.10 Lighting Tree

The root of the tree is the origin in space of the initial ray (view point). At the completion of the rendering process this node will contain the final color value for the initial ray. The intensities returned at each point are the intensities of the three primary colors (red, blue, and green). An incident ray is generated from the root node in the desired sampling direction. If this ray, or any other ray, does not intersect an object, a background intensity value is returned to the node where the ray originated. If the incident ray intersects a surface, one or more secondary rays are generated. One set of secondary rays are the shadow rays which are generated from the intersection point to each light source. These rays return a pass-fail result indicating if there is an object between the light source and the intersection point. To generate the remaining secondary rays (if any) and to calculate the shading values, the normal to the surface must be determined. The calculations required to find the normal to the surface are dependent on the type of surface (polygon, quadric, etc.) and any smooth shading or bump mapping algorithm used. If the surface is nonreflective and opaque, the intensity of the surface is calculated using the surface normal and the desired lighting model. This intensity value is returned to the parent node. If global lighting

information is required, as is the case for reflective and transparent surfaces, secondary rays are generated in the appropriate directions. These secondary rays are traced in the same manner as the incident ray. When the results from the rays are returned, they are used to complete the lighting calculation. The intensity values are returned to the parent node. The lighting tree is an integral part of the ray tracing process and is generated and evaluated recursively.

An examination of Figure 2.10 reveals that the total number of rays due to a single incident ray can grow exponentially with the depth of the tree depending on the complexity of the scene being rendered. Smaller virtual lighting trees require fewer rays to be traced thus improving the speed of the rendering process. The key to limiting the size of the tree is to reduce the number of reflective and transparent rays traced. As mentioned earlier, rays are not traced if the surface has no reflective or transparent component. Physical observations of real scenes which contain many reflective and transparent objects, suggests that after about four or five reflections or about seven transparent objects there is little more additional information to be obtained. This is mainly due to the distortion and reduction of light intensity. A ray tracing system can take advantage of this observation by limiting the depth the lighting tree. As the lighting tree is expanded, the percentage of possible contribution to the intensity of the root node is passed with each secondary ray. These values are obtained from the incoming percentage and the coefficients for the reflective or transparent lighting components. If the incident ray strikes a surface which is 70% reflective and 30% transparent then the value .7 is passed with the reflective ray. If the reflective ray strikes a surface which is 50% reflective then the value of .35 is passed with the next reflective ray. Once this value reaches some minimum amount, or the number of rays reaches some maximum depth, no further secondary rays are generated.

2.3.5. Smooth Shading

A smooth surface can be simulated from a mesh of polygons using some type of curved surface simulation algorithm. The first such method was proposed by Gouraud [20] and has become known as Gouraud shading. The first step in Gouraud shading is to set the normal values at a node of a polygon to the average of the normals of all the polygons that contain this node. This procedure is shown in Figure 2.11a.

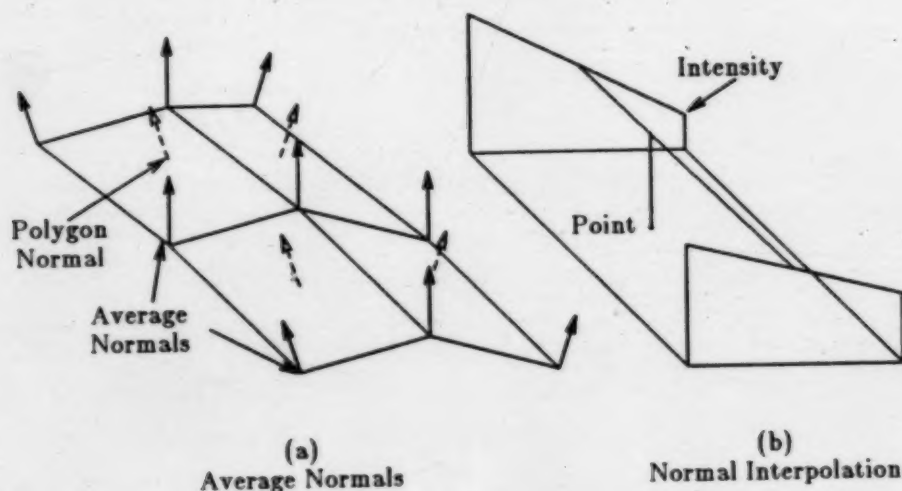


Figure 2.11 Smooth Shading

To determine the intensity of an arbitrary point on the polygon, Gouraud first determines the intensity at the nodes of the polygon using the average normals and the desired lighting model. The intensity of an arbitrary point on the polygon is a linear interpolation of the intensities at the nodes as demonstrated in Figure 2.11b.

Gouraud's model requires the light source to be sufficiently far from the surface that the angle between the light source and the surface of the polygon can be considered a constant at any point on the surface. Phong [7] improved on Gouraud Shading by interpolating normals rather than intensities at the intersection point to overcome this problem. Duff [16] presented work that generalizes the determination of the

normal at an arbitrary point on the polygon based on the normals at the nodes and a predefined function which need not be linear.

The values obtained using Phong or Gouraud shading are dependent on the orientation of the polygon in space. As the polygon is rotated, the pair of edges used to interpolate the value at a single point may change causing noticeable discontinuities in the lighting intensity of the surface. Further, the interpolated curvature of the polygon in a mesh can change abruptly across the surface resulting in single frame shading discontinuities. These effects are most apparent if the polygon has more than four edges.

A method of smooth shading that uses a normal located at the approximate center of the polygon, in addition to the normals stored at the nodes was constructed as part of this thesis. The value of this center normal is the average of the normals at the nodes of the polygon. The normal at an arbitrary point on the polygon is a linear average between the center normal and the normal at the edge of the polygon which falls on a straight line between the center normal and the intersection point. This technique is shown in Figure 2.12

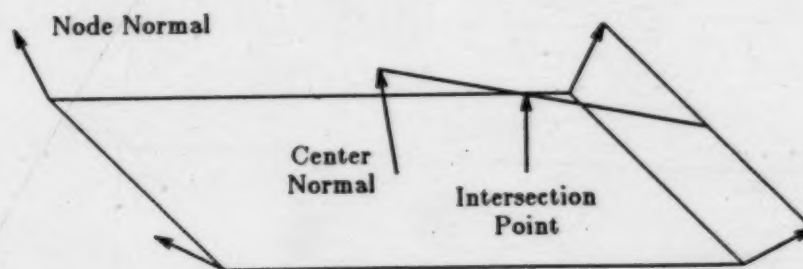


Figure 2.12 Smooth Shading With Center Normal

The center normal is used in the calculation of the normal at all points on the surface of the polygon. Since the center normal is the average of the normal at each node of the polygon, the curvature of the polygon is more evenly distributed across the

surface than with Gouraud or Phong shading. Further, the averaging algorithm is independent of the orientation of the polygon in space thus eliminating the abrupt intensity changes (Mach Bands) which occur during an animation sequence using Gouraud or Phong shading techniques.

2.3.8. Texturing and Bump Mapping

Texture and bump mapping are two powerful graphics techniques that can add considerable visual complexity to an image at a low computational cost [3]. Texture mapping involves modifying the intensity of a point on the surface, whereas bump mapping involves changing the surface normal. Each surface in the scene may be texture and bump mapped based on a predefined mapping function. The function used for mapping can be analytical, such as the first few terms of a Fourier series, or an array of data values. Two dimensional mapping values are usually a function of the position of the intersection point relative to the boundaries of the surface being mapped.

Texture mapping is applied at each node of the lighting tree when the surface found at that node has been specified as being textured mapped. The lighting intensity of the node is first determined in the regular fashion described earlier. For two dimensional texture mapping, the location of the point of intersection on a surface relative to the surface boundaries is often obtained from the values of parametric variables. If the surface initially was described parametrically, these values are already available. If not, and the surface is equivalent to a patch, the parametric values can be obtained in a similar fashion to the method Ullner used to find the u and v values from the corners of a polygon [40]. The parametric values are used to index into a two dimensional array or are used in some other type of function to obtain the mapping value. The mapping value is used to modify the intensity values at the node, usually as a weighting function. The mapping array is often an image that has been previ-

ously rendered or digitized with a camera. The image in the mapping array will appear as though it was painted onto the mapped surface. If the map has color information then there is one set of mapping values for each of the three primary colors.

Bump mapping is performed in a similar fashion as texture mapping except the normal to the surface is modified based on the mapping function. The mapping value is used to rotate the normal to the surface. This affects the direction of the secondary rays and the intensity of the surface. The resulting effect in the image appears as a rough or bumped surface finish, such as a tiled wall.

Three dimensional texture mapping has recently been proposed by Peachey [29]. The texture function is applied based only on the position of the intersection point in three space. A three dimensional map is used for this purpose. If the surface is cut or molded, the texture will correctly run through the material, such as for wood and marble.

2.4. Distributed Ray Tracing

Distributed ray tracing was first introduced by Cook, Porter and Carpenter [12] in 1984. Distributed ray tracing spatially or temporally distributes rays to obtain a different range of information from the scene then would be determined normally. The technique effectively renders lighting effects such as motion blur and penumbra shadows.

The principle behind distributed ray tracing is to modify the usual direction of a ray, based on the lighting effect desired. Often, more rays than the number required for sampling are generated by the distributed ray tracing algorithm. These extra rays are incident ray or secondary rays. The extra rays are traced to different parts of the scene, as is the case for penumbra shadows, or during different times, as is the case for motion blur.

One application of this algorithm is to obtain penumbra shadows. Rather than tracing one ray to the center of each light source, a group of rays are sent to different parts of each spherical light source. Figure 2.13 shows two of the secondary rays used to test for shadows from a point on the surface.

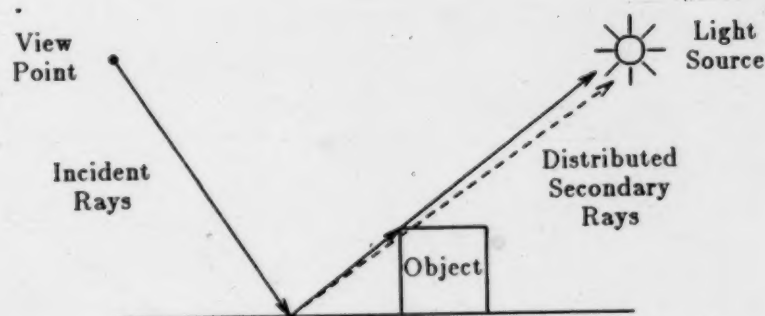


Figure 2.13 Distributed Ray Tracing

Along the edges of the shadow of an object, some of the secondary rays miss the object and strike the light source while others are blocked. When the rays are averaged to obtain the final pixel value, the shadow will appear lighter around the edges of the object resulting in the penumbra effect. The degree of this effect is dependent on the placement of the objects in the scene and the size and location of the light source. Further examples and details can be obtained from the paper by Cook, Porter, and Carpenter [12] on distributed ray tracing.

2.5. Beam and Cone Tracing

Beam tracing is a variation of ray tracing which takes advantage of the spatial coherence of rays. This technique introduced by Heckbert and Hanrahan [24] builds a beam tree while rendering the image. The initial beam is the viewing pyramid. The first node of the beam tree contains the intersection of the viewing pyramid with the objects in the scene. Secondary beams are created for reflective and refractive surfaces. The data primitives in the scene are limited to polygons, and the secondary

beams are the shape of the intersection of the beam and the polygonal surface from which they originate. A linear transformation is used to modify the data in the scene so that the secondary beam views the world as a reflection or transparent image viewed from the surface. The transformation of the scene for transmitted beams must be approximated with a linear transformation. Beam tracing has limited application due to the use of only polygons and the inability to include all mapping techniques.

Cone tracing was introduced by Amanatides [1] in 1984. Rather than tracing a ray from the eye through the pixels on the display, a cone is traced that covers most of the area of the pixel. This procedure reduces the aliasing problem that occurs with point sampling techniques by sampling the entire pixel area. The intersection calculation of cones with objects can be quite complex. Cones will fragment and distort during reflections and refractions, further complicating the calculations.

2.8. Data Hierarchies

The ray tracing algorithm, along with other hidden surface algorithms, can be viewed as a sorting problem. This approach to analyzing hidden surface algorithms was presented by Sutherland, Schumaker, and Sproull [38] along with a taxonomy of ten hidden surface algorithms. The key to reducing the time to sort data during rendering is to take advantage of several types of coherence in the image at several stages along the rendering pipeline. The efficiency of the rendering procedure can be further improved if some of this sorting can be done before the rendering process begins. This is a major function of data hierarchies.

Earlier work by Clark [9] created a data hierarchy that was designed to improve the efficiency of the clipping process. Clark's hierarchy allowed objects outside the viewing pyramid to be quickly eliminated from consideration and eliminated sub-pixel detail from being rendered. Further work by Rubin and Whitted [36] examined data hierarchies which would eliminate large portions of the image from consideration

during the ray tracing process. These techniques were improved and expanded by Glassner [18], Weghorst, Hooper, and Greenburg [41], and Cleary and Wyvill [10] in order to more efficiently construct and traverse these structures. The two main techniques used to subdivide the data into a hierarchy are based on the volume of the scene (volume hierarchies) or on the local coherence of primitives (object hierarchies).

Several factors affect the application and effectiveness of data hierarchies. The initial consideration is the time and space required to construct the hierarchy. If the rendering package is part of an animation system, as is often the case, the robustness of the hierarchy is of concern. The hierarchy is robust if it requires little modification between frames of an animation sequence. Some hierarchies must be rebuilt if an object is moved or modified. A second consideration is the speed at which the hierarchy can be traversed during rendering under many scene conditions. Rendering efficiency has been the major concern in the design of a data hierarchy. The object and volume based data hierarchies are now examined.

2.8.1. Object Hierarchies

The construction of the object hierarchy is considered from the bottom up. The simplest objects are defined as a collection of primitives at the lowest object nodes of the hierarchy. Simple objects are those which are not usually decomposed. Examples of low level objects are bolts, table legs, a leaf on a plant, and a handle on a drawer. This is the only place in the object hierarchy where data primitives are stored. More complex objects are defined as collections of several simpler objects. Examples of such objects are trees, desks, and cars. This building process continues until all the scene is contained under a single node of the hierarchy (the root node). Object hierarchies are graphs where common objects, such as a window in an office building, is shared between many higher level objects. To be able to share objects, each object node contains a transformation matrix for each child. Figure 2.14 shows a partial object hierar-

chy of a car. The windows used in the doors of the car are identical but in different locations. Only a single window node is needed as they are translated to the proper positions for each door by the transformation matrix at the parent node.

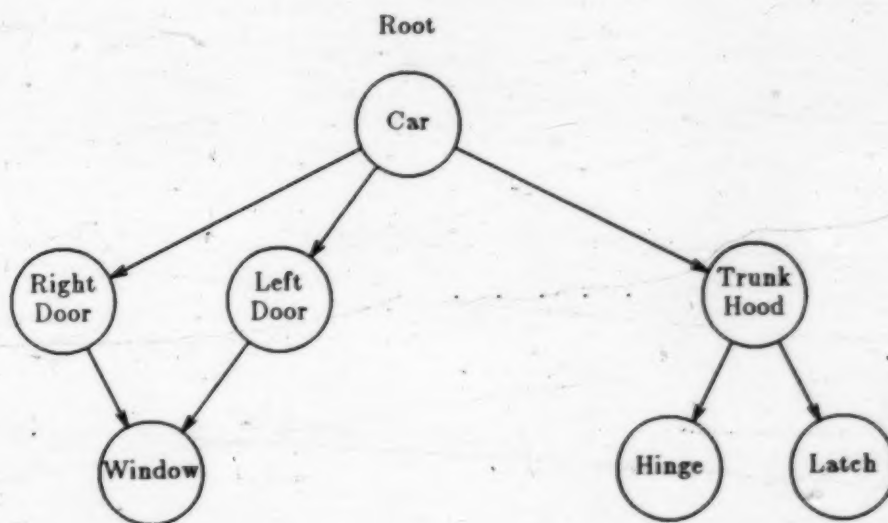


Figure 2.14 Object Hierarchy

To determine the first object the ray strikes, the root node of the object hierarchy is examined first. Before the rendering process is started, bounding volumes are created and stored at each node of the object hierarchy. These volumes enclose all the data below the node in the hierarchy. When a node is visited by the rendering system, the current ray is tested for intersection with the bounding volumes at the node. If the ray fails to intersect any one of the bounding volumes at a node then the ray cannot possibly intersect any of the data within the bounding volume, and the hierarchy below this node can be pruned. A further discussion of bounding volumes is presented later.

If the ray intersects the bounding volume at a node, each child of the node must be examined. This process continues until all the nodes have been visited or culled. In

many cases, only one object node which contains data primitives will pass the bounding volume test. Each data primitive in such a node, is examined to find the primitive which intersects the ray and is closest to the ray's origin. Once this primitive is found, the traversal operation is complete. If several objects in the scene are placed one in front of each other, then the ray may intersect the bounding volumes for each. In such cases, there is more than one object node which contains data and passes the bounding volume test. In this case each data primitive in each of these nodes must be tested to determine the single primitive which is the closest.

Part of the animation sequence involves moving objects within the scene or adding and deleting objects between frames. To facilitate object motion an extra transformation matrix is stored at each object node of the hierarchy. This matrix is applied to the node at which it is stored and to all descendants. During rendering, a composite transformation matrix is kept with the ray. This matrix is initially set to the identity matrix. When each node is visited the transformation matrix stored at the node is concatenated with the matrix stored with the ray. As the ray progresses down the tree it accumulates all the transformations from the higher level objects. When bounding volumes or data are tested, the composite transformation matrix is applied first. The matrix can be applied to the data or its inverse applied to the ray. The result is identical, and in the later case, may result in simpler calculations. The power of this structure can be demonstrated with a simple example using the car shown in Figure 2.14. If the car is to be moved five feet forward, the animations system would update the transformation matrix at the node marked car. During traversal, this matrix is applied to all descendants of the node and as such, each piece of the car will move five feet forward. Finally, objects are added to, or deleted from, the hierarchy by simply making or breaking the appropriate links between nodes.

The construction of the object hierarchy is done manually. The designer models the physical relationship of objects using the appropriate parent-child relationships.

The traversal of the object hierarchy is summarized in the algorithm shown in Figure 2.15. The routine *Traverse* calls the routine *Object_Traverse* for each ray.

```

Traverse() {
    For( Each Ray in Image ) {
        Current_Primitive = NULL; Ray.Matrix = Identity;
        Object_Traverse(Root, Ray);
    } }

Object_Traverse(Obj, Ray) {
    Ray.Matrix = Ray.Matrix * Node.Matrix;
    If( Ray Strikes Node.Bounding_Volumes ) {
        For( Each Child of Obj ) {
            If( Children are Object Nodes ) {
                Ray.Matrix = Ray.Matrix * Node.Child.Matrix;
                Object_Traverse(Obj, Ray);
            } Else {
                For( Each Primitive in Child ) {
                    If( Ray Intersects Primitive ) And
                        ( Primitive is Closer Than Current_Primitive )
                        Current_Primitive = Primitive;
                } } } }
    Return Current_Primitive;
}

```

Figure 2.15 Object Hierarchy Traversal

2.6.1.1. Bounding Volumes

Bounding volumes are used to cull as much of the hierarchy as possible. Bounding volumes that tightly fit the objects they contain are more effective for pruning the hierarchy than loose fitting volumes, provided they are not too difficult to test for intersection. Simple volumes, such as spheres and boxes, often fit loosely around the data but are simple to test for ray intersection. Such volumes are often used as preliminary tests. More elaborate bounding volumes such as ellipsoids can be used, but their advantage is dependent on the time required to intersect the volume relative to the time required to traverse the hierarchical structure below the node. Weghorst, Hooper, and Greenburg [41] present an analysis of the effect of the fit of a bounding volume to the data on the efficiency of the traversal algorithm.

2.6.2. Volume Hierarchies

Volume based hierarchies divide the entire volume in the scene into recursively smaller subvolumes. The scene volume can be divided into equal area subvolumes or divided into unequal volumes based on the local complexity of the image. The advantage of volume based hierarchies is the known ordering among volumes. Volumes are visited in the order the ray intersects them during rendering.

All the space in the scene is directly represented in the volume hierarchy. This is in contrast to object hierarchies where only the space containing objects is represented. Figure 2.16 shows a scene volume that has been divided into a two dimensional array of subvolumes.

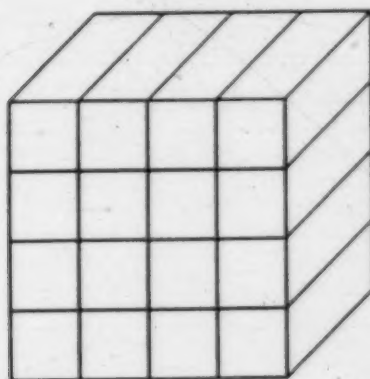


Figure 2.16 Regular Volume Based Hierarchy

Construction of the volume hierarchy begins by examining the data to find a minimal box, aligned with the coordinate axis, which contains all the objects in the scene. If the number of primitives in this volume exceeds some limit (usually 4 to 30) the volume is subdivided, often into an oct-tree structure. The initial volume is divided into eight subvolumes about the center of the original volume with the new volumes aligned with the coordinate axis. This procedure creates eight equal subvolumes, each of which contains only the primitives in the scene that are in their part of the original volume. Each of the subvolumes then are individually examined. If the volume requires subdivision, the same procedure is applied. With the volumes aligned with the coordinate axis, it is relatively easy to clip and split the primitives that occupy two or more volumes. The subdivision process is repeated till a minimum volume size is obtained. The subdivision process can be designed such that each volume is divided if there is any volume which contains an excessive number of primitives [10] [40]. This results in a regular structure where each volume is of the same size. Such hierarchies are referred to as regular volume hierarchies in this thesis. Alternatively, the volume division could be based on the number of primitives contained in each volume independently [18]. This type of volume hierarchy is referred to

as an adaptive volume hierarchy in this thesis. A construction tree is built by the adaptive volume hierarchy to record how the hierarchy was created. The advantages and disadvantage of each method will become apparent during the discussion of the traversal stage of the rendering system.

Traversing the volume hierarchy begins by determining the volume where the ray originates or the first volume the ray intersects. If a regular hierarchal structure is used, the initial volume is directly calculated from the description of the ray, size of the hierarchy, and the degree of subdivision. The irregular hierarchal structure, introduced by Glassner [18], uses a hash table and a construction tree to find the initial volume. The ID of the volume the ray originates is found using the construction tree. The data primitives in the volume are accessed using the hash table.

The data primitives in the initial volume are tested for intersection with the ray. If an intersecting primitive is found, the traversal operation is complete and the closest primitive in the volume returned. It is the ordering of the volumes that guarantees that no other primitive which intersects the ray can be closer. If the ray does not intersect a primitive in the original volume, the next volume the ray enters is determined and examined. The next volume the ray enters is determined directly when using a regular volume hierarchy. The adaptive volume hierarchy must first find a point in the next volume and use the construction tree to obtain the ID of the volume. The hash table is used to access the data stored in this volume. This overhead is the major cost of using the adaptive volume hierarchy. The cost of using a regular hierarchy is the space needed to save pointers to empty volumes and the time required to traverse these empty volumes during rendering. Under optimal conditions, a primitive is found in the first volume visited. This results in a constant time algorithm. These algorithms are discussed further in Chapter 4.


```

Build_Hierarchy(Data){
    Create Initial Volume_List With One Volume Contain Entire Scene;
    Set Min_Volume; Set Max_Primitives;
    Build_Volume(Volume_List);
}

Build_Volume(Volume_List){
    If( ( Any Volume Contains More Than Max_Primitive ) And
        ( Volume Size > Min_Volume ) ){
        For( Each Volume ){
            Clip About Volume's Center into Eight Sub-Volumes;
            Replace Volume With Eight New Volumes on the Volume List;
        }
        Build_Volume(Volume_List);
    } }

Traverse(){
    For( Each Ray ){
        While( No Intersecting Primitive Has Been Found ){
            Find Next Volume Ray Enters;
            Find Closest Intersecting Primitive In That Volume;
        } } }

```

Figure 2.17 Volume Hierarchy Construction and Traversal

The algorithms used to create and traverse the regular volume hierarchy are summarized in Figure 2.17. The initial data is passed to the routine *Build_Hierarchy* where the original volume and subdivision termination criteria are set. The data are then recursively subdivided by the routine *Build_Volume* until the termination criteria

is satisfied. The routine *Traverse* is used to traverse the hierarchy.

If objects are moved between frames of an animation sequence, the data in the volume based hierarchy must be, resectioned, moved, added, or deleted. This procedure will often result in more empty volumes and some over-crowded volumes. As a result, the entire volume hierarchy is usually rebuilt between each frame.

In general, object hierarchies are more robust than volume hierarchies. During an animation sequence, the transformation matrices at each node in the object hierarchy provide sufficient flexibility so that the data hierarchy can be used as is. Volume hierarchies are usually rebuilt whenever an object moves from one volume to the next.

Volume hierarchies are generally more efficient to traverse than object hierarchies. Only the volumes the ray enters are examined. These volumes are examined in such an order that once an intersection is found in a volume, the search can be terminated. Object hierarchies require a graph to be traversed. If a ray intersects a bounding volume at one node, all children of this node must be expanded.

2.7. Multiprocessor Ray Tracing

The regular volume hierarchy described earlier has been used by Cleary [10] and Ullner [40] to define a parallel ray tracing algorithm. Each researcher has proposed an architecture for implementing their algorithm. Simulations performed by Cleary show that little is gained by using a three dimensional array of processors over a two dimensional array. Further, the three dimensional array presents difficult implementation problems. A two dimensional parallel ray tracing architecture is examined here.

Figure 2.18 shows the organization of a two dimensional array of processors. A host computer with a disc and a frame buffer is connected to one of the processors to send and received the data. The structure is based on a regular volume hierarchy. Each processor is assigned one of the subvolumes shown in Figure 2.16.

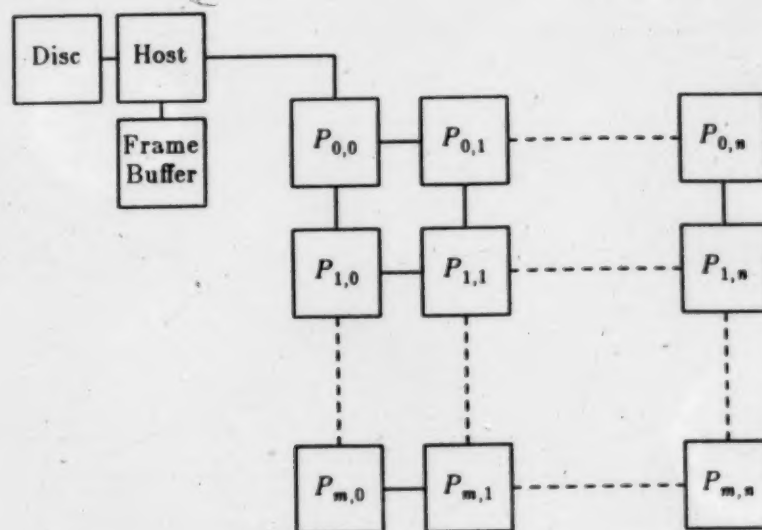


Figure 2.18 Parallel Ray Tracing Array

Each node is initially loaded with the rendering code by a serial line or the code is passed from one processor to the next using the communications lines between each processor (see Figure 2.18). The data are loaded into the first processing node by the host. This processor clips the data against its assigned subvolume and passes all the data that falls to the right of its subvolume to the processor on its right. The part of the remaining data that falls below the assigned subvolume is passed to the processor below. The data left in the processor is the only data that falls in the assigned subvolume of that processor. This process is repeated at each node until all the data has been distributed.

Ideally the view is located such that an equal number of incident rays enters each subvolume from the front or back faces. Each processor is then required to trace the same number of incident rays which helps equalize the processing load. Each processor then begins testing the initial rays against the primitives stored at its node. If a ray fails to intersect any primitive in a node, the ray must exit the assigned subvolume of the processor through a side of the volume. If the ray passes through a side that is

connected to a neighboring processor then the ray may still intersect a primitive and this ray is passed to the neighboring processor for intersection testing. If a ray strikes a surface, secondary rays are generated. These secondary rays are intersected with each primitive in the node and processed in the same fashion as the initial rays. A message system is used to keep track of rays, and is responsible for passing final pixel values back to the host.

The processors that contain the light sources or complex parts of the image are responsible for more work than many of the other nodes. Also, if the view is such that initial rays enter only a few processors, many processors could remain idle throughout the entire rendering process. Ullner [40] discusses some methods of reducing this problem but the results are not effective in the general case.

2.8. Summary

This chapter presented several algorithms, techniques, and architectures used for generating realistic computer images. The previous work on lighting models, data structures, and parallel ray tracing forms the foundation on which the research in Chapters 3 and 4 is based. The other topics discussed in this chapter outline the processing and data requirements of the ray tracing task. Any improvements to the ray tracing system must carefully consider these requirements to obtain effective results.

Chapter 3

Lighting Models

3.1. Introduction

Most scenes encountered in the real world consist primarily of surfaces with large diffuse lighting components [19]. Examples of such surfaces include walls, cloth, bricks, rocks, and paper. If two diffuse surfaces are brought near each other one can often see some of the color from one surface in the other. This occurs because a diffuse surface acts as a secondary light source, reflecting some of the light that impinges upon it. This light in turn illuminates the second object.

A lighting model is developed in this chapter that includes lighting contributions from global light sources in the diffuse lighting calculation for a surface. The diffuse contributions from the primary and secondary light sources are considered separately. Primary light sources are modeled as point light sources and their contribution to the diffuse component determined using Lambert's cosine law. The contributions from the secondary light sources to the diffuse component are obtained with a ray tracing algorithm. It is shown that this model is more efficient and more general than existing proposals.

An algorithm for the efficient implementation of this model is presented. This method uses a distributed ray tracing algorithm to sample global light sources. Distributed ray tracing reduces the number of rays that must be traced to obtain an image with tolerable sampling errors.

Diffuse lighting effects play a major role in natural scenes. The effects range from soft shadows and color bleeding between surfaces, to projected light from a mirror or through a glass lens. The effectiveness of the new lighting model using the suggested implementation techniques is discussed and example images are presented.

3.2. Improved Lighting Model

The intensity of a point on a surface is the integral of the light energy from all directions about the point on the surface, times a weighing function. In nature the weighing function is dependent on many factors including the location of the light source relative to the surface normal, the frequency of the light, the direction the surface is viewed from, and the physical composition of the surface material. A general and complete lighting model that is computationally feasible does not yet exist. Current lighting models differ in how they include various weighting factors or the way they account for light from different directions around the surface. The new lighting model presented in this chapter obtains global lighting contributions for the diffuse component of a surface in a new manner.

The lighting models presented in Section 2.2, and the new lighting model presented in this chapter, use a weighting function which is dependent on only a few of the possible weighting parameters. The weighting function is considered to be dependent only on the location of the viewer and light source relative to the normal of the surface, and the color of the surface and the light sources. The color of all objects and light sources is specified by the relative intensities of the primary additive colors (red, green, and blue). All color effects are limited to the interaction of these values.

The new lighting model views a surface as possessing three main lighting components. Equation 3.1 shows the three main components of this model.

$$I_S = I_D + I_R + I_T \quad 3.1$$

where,

I_S = The intensity of the surface.

I_D = The diffuse component.

I_R = The reflective lighting component.

I_T = The transmitted lighting component.

The first component, known as the diffuse term, accounts for diffuse reflections from a surface due to all light sources. The intensity of this light is weighted by the color of the surface and the color of the light sources. The second component, known as the reflective term, is also a reflective component. This term is independent of the color of the surface, but is weighted by the color of the light sources. The third term, known as the transmitted or transparent term, accounts for light that is transmitted through the object. The light in this term is weighted by the color of the interior of the object and the color of the light sources. The division of the reflective term into a diffuse and specular component is exploited later in this chapter to improve the overall effectiveness of the lighting model.

The new lighting model obtains the required lighting information from the description of the scene using a ray tracing algorithm. The lighting information is collected into a lighting tree and the lighting model applied to each node. The ray tracing algorithm and the application of lighting trees is described in Section 2.3.

The intensity of any given point on a surface is a function of the light falling on the point from all directions. The reflective terms account for light from above the surface while the transmitted term accounts for light from below the surface. The contributions from these light sources are obtained by tracing rays from a point on the surface in many different directions. Equation 3.2 is the new lighting model using ray traced terms.

$$I = \sum_{i=1}^{N_{RD}} R_{D_i} W_{R_{D_i}} + \sum_{i=1}^{N_{RS}} R_{S_i} W_{R_{S_i}} + \sum_{i=1}^{N_T} T_i W_{T_i} \quad 3.2$$

where,

I = The intensity of a point on the surface.

N_{RD} = The number of diffuse reflective directions tested about a point.

R_{D_i} = The lighting value found in the i^{th} diffuse reflective direction.

$W_{R_{D_i}}$ = The weight of light energy found in the i^{th} diffuse reflective direction.

N_{R_S} = The number of specular reflective directions tested about a point.

R_{S_i} = The lighting value found in the i^{th} specular reflective direction.

$W_{R_{S_i}}$ = The weight of light energy found in the i^{th} specular reflective direction.

N_T = The number of transmitted directions tested about a point.

T_i = The lighting value found in the i^{th} transmitted direction.

W_{T_i} = The weight of light energy found in the i^{th} transmitted direction.

The sum of all the diffuse weights W_{D_i} is the diffuse coefficient of the surface. The sum of all the reflective weights W_{S_i} is the reflective coefficient of the surface. The sum of all the transparent weights W_{T_i} is the transparent coefficient of the surface. The sum of the diffuse, reflective, and transparent coefficients is less than or equal to one if the surface does not emit light.

Diffuse lighting is conceptually simple, but the interaction of diffuse light between several objects can result in complex lighting effects. The basic properties of diffuse lighting were introduced in Section 2.2. An important addition to the basic lighting model introduced in this chapter is the definition of a feasible and effective method of implementation. The lighting model described by Equation 3.2 is modified by considering the lighting effects from secondary and primary light sources separately. This technique ray traces only the contributions from secondary light sources and uses conventional models to include the lighting contributions from the primary light sources. The conventional models, such as Phong's Lighting Model [7] are very efficient in the evaluation of primary light sources. Considering the primary and secondary light sources separately is a particularly useful technique when applied to the diffuse component, but does have applications to the reflective and transmitted lighting components of a surface. The application of this idea to the diffuse component is exam-

ined here.

The diffuse component of Equation 3.2 is divided into two parts. Equation 3.3 is the resulting model.

$$I = K_D \sum_{i=1}^{N_{PL}} I_{L_i} \cos(\theta_i) + K_D \sum_{i=1}^{N_{RD}} R_{D_i} W_{R_{D_i}} + \sum_{i=1}^{N_{RS}} R_{S_i} W_{R_{S_i}} + \sum_{i=1}^{N_T} T_i W_T \quad 3.3$$

where,

N_{PL} = The number of point light sources.

θ_i = The angle between the surface normal and the i^{th} light source.

I_{L_i} = Intensity of the i^{th} light source.

K_D = Diffuse lighting coefficient of the surface.

The diffuse term of Equation 3.2 is replaced by the first and second terms shown in Equation 3.3. The first term of Equation 3.3 models the diffuse lighting contributions from the primary light sources, while the second term models diffuse lighting contributions from secondary light sources. While the second term of Equation 3.2 and 3.3 appear similar, the value of the variable N_{RD} is substantially smaller in the new equation to produce the same image because primary light sources are not included in the term. This effect is discussed in detail in Section 3.5. The value of K_D was factored from the diffuse weights in the second term so that it could be insured that the sum of the weights of the ray traced light sources, $W_{R_{D_i}}$, and the point light source, $\cos(\theta_i)$, is equal to one.

The new lighting model differs in three ways from the models presented in Section 2.2. First, the contributions from individual secondary light sources, independent of the type of light source (diffuse, reflective, etc.), are included in the diffuse lighting calculations. Second, an arbitrary number of secondary rays can be generated from any incident ray depending on the geometric and lighting complexity of the scene. This

second feature is required to accurately account for contributions from secondary light sources in the diffuse component of a surface, and allows for gloss and other lighting features in the reflective and transmitted components. Finally, each component of the lighting model is modeled in a similar way. This model, while including more lighting effects than previous models, does not include all lighting effects. The most notable effects missing from this model are wave interference, true spectral effects, polarization, and object phosphorescence.

All of the lighting models described in Section 2.2 are viewed as special cases of Equation 3.3. For example, to obtain Whitted's lighting model first assume that all surfaces are perfect reflectors and transmitters of light. Perfect reflectors and transmitters require only a single secondary ray to be traced to accurately account for secondary lighting contributions. The values of N_{R_S} and N_T for the reflective and the transmitted terms of Equation 3.3 are both one. The reflective weight $W_{R_S_1}$ is equal to the reflective coefficient of the surface and the transmitted weight W_{T_1} is equal to the transparent coefficient of the surface. Whitted also approximated the diffuse lighting term by only considering the contributions from primary point light sources and used an ambient lighting factor to account for the diffuse light from all secondary light sources. The second diffuse lighting term of Equation 3.3 is replaced with an ambient constant term. Using these assumptions and simplifications, Equation 3.3 reduces to the following.

$$I = K_D \sum_{i=1}^{N_{PL}} I_{L_i} \cos(\theta_i) + I_A + R_S K_R + T_S K_T \quad 3.4$$

where,

I_A = Ambient lighting value of the surface.

K_R = Reflective lighting coefficient of the surface.

K_T = Transparent lighting coefficient of the surface.

Equation 3.4 is identical to Whitted's lighting model described in Section 2.2. Phong's lighting model can be obtained directly from Whitted's model in a similar manner.

Equation 3.3 requires that each object in the image be divided into primary light sources and secondary light sources. Each of the primary light sources are approximated with one or more point light sources. Each of the point light sources are modeled using Lambert's cosine law. This insures that the primary light sources are not sampled by the ray traced terms. The effects of the secondary light sources on the intensity of a surface is more pronounced if the secondary light source is close to the object. The effect of distant objects is minimal. By reducing the strength of a diffuse ray based on the distance it travels before striking an object the effect of distance objects is minimized. This has the added benefit of reducing the size of the lighting tree. The loss of the contribution of the distant objects is compensated by the addition of an ambient lighting value. The size of this parameter is usually less than five percent of the total diffuse component.

The capabilities and effects of Equation 3.3 are best understood by examining its ability to render images of varying lighting complexities. The discussion of such images will highlight the effect of each term and variable in the equation on the images. This discussion, and the images produced using this algorithm, are presented in Section 3.5. Before presenting and examining these images, the selection of directions for the diffuse secondary rays and several implementation techniques for the model are presented.

3.3. Secondary Ray Distribution

The information collected by a ray is divided into geometric and lighting information. Geometric information is concerned with the location of different surfaces in the scene. Lighting information is concerned with the location and characteristics of the light sources in the scene. Sampling errors result when an insufficient lighting or geometric information is obtained in a particular area of the scene. Dynamic sampling techniques are used to obtain greater information in complex areas of the image but these methods require some initial set of rays to be traced to determine the areas where further sampling is necessary. Each ray of this initial set should be directed such that each obtains an equal amount of information. The distribution of diffuse rays, based on the lighting information obtained, is examined first.

Lambert's cosine law relates the amount of light energy received by the surface relative to the location of the light source. Equation 3.5 is Lambert's cosine law.

$$I_D = I \cos(\theta) \quad 3.5$$

where,

θ = Angle between the surface normal and light source.

I = Intensity of the light source.

I_D = Diffuse intensity of the surface

Equation 3.5 shows that rays close to the surface normal pass more energy to the surface. The vector field of this energy results in a sphere resting on the point of intersection on the surface of the object. A two dimensional cross section of this sphere is shown in Figure 3.1a. Each of the diffuse rays can be viewed as passing through the center of a spherical cone. This cone is located inside the sphere with its tip at the south pole of the sphere and terminating at the surface of the sphere. Each cone should be of equal volume so that each ray represents an equal amount of light energy.

This requirement tends to cluster rays near the surface normal.

The initial set of rays also can be distributed such that they obtain equal amounts of geometric information. Each ray should sample the same amount of scene volume. That is, each of the diffuse rays should have the same solid angle. The geometric requirements result in a different set of initial ray directions than the ones defined by lighting requirements for any given number of initial rays. The determination of the optimal distribution proved to be a non-trivial mathematical problem and is beyond the scope of this thesis. An alternative solution is examined next.

The weight passed with each diffuse ray is determined by the amount of lighting information the ray has been assigned. The lighting tree is trimmed when the weight of a ray falls below some minimum value as described in Section 2.3. To reduce the height of the tree, each diffuse ray is directed such that each has the same initial weight. The direction of the initial set of diffuse rays is primarily dependent on equalizing the lighting information. There are many possible patterns where each diffuse ray has the same weight. A pattern must be chosen so the geometric information collected by each ray is similar.

In order to insure that the rays are evenly directed into the scene several simplifying assumptions were made. First, it was assumed that the rays would be distributed in several unique bands. These bands, shown in Figure 3.1b, are divided by dashed lines. Rays are directed through each band but rays in different bands are never in the same plane.

Assume that there are 25 secondary rays available for redirection. By experimentation it was determined that five bands, with the first band containing a single ray and each other band containing six rays each, produced satisfactory results. Modifications to this scheme, such as five bands each containing five rays, had little effect on the final image. This indicates the actual distribution was not critical and any

reasonable geometric distribution can be used.

The angles between the normal and all rays in a given band are the same. Within a band the rays are evenly distributed around the surface normal and each band is offset from the next such that no two rays are directly above each other. Figure 3.1c shows a top view of the rays for two levels, each containing six rays, leaving the point on the surface. The volume of a given band can be represented by the difference of two spherical cones as shown in Figure 3.1b. The angle between the normal and the rays in each band was determined such that all rays in each band represent the same lighting intensity. If there were nine diffuse rays in all, and one band contained three of these rays, the volume of this band would be one third of the total volume. An initial set of N unit vectors are calculated once at the start of the rendering session, where N is the number of initial secondary diffuse rays desired. The set of unit vectors are determined relative to an assumed surface normal on the $+y$ axis. These unit vectors must be rotated so that the $+y$ axis coincides with the normal to the surface at any particular intersection point. The rotated set of unit vectors is used as the secondary diffuse rays.

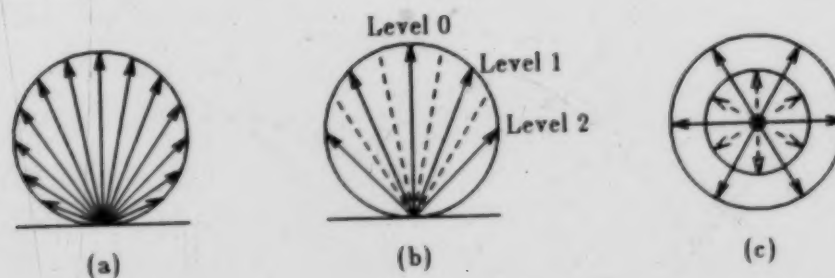


Figure 3.1 Secondary Ray Distribution

During rendering, if the difference between two secondary diffuse rays is larger than a limit which is a function of the weight of the incoming ray, an extra ray is traced between them.

This entire technique can be applied to the highlights found on glossy objects. The effects from the primary light sources can be modeled with a cosine function raised to a power and the contributions from secondary light sources ray traced using an appropriate sampling pattern.

3.4. Implementation

The following implementation techniques are designed to reduce the time required to render a particular lighting effect, thus enhancing the effectiveness of the algorithm. Complex lighting effects require more rays to be traced to obtain an image with tolerable sampling errors. The following techniques reduce the number of rays generated for a particular quality scene.

The ray minimization techniques described in Chapter 2 involved not tracing secondary rays which contribute an insignificant amount to the intensity of the initial incident ray. This includes not tracing reflective rays from matte surfaces (no reflective lighting component), not tracing transmitted rays on opaque objects, and not tracing diffuse ray from non diffuse surfaces. These basic techniques also eliminates rays which have undergone multiple reflections, refractions, and diffuse reflections.

A second technique discussed in Section 2.4 involved insuring each ray traced obtains the most information possible. If a surface has both diffuse and reflective components, the information obtained by tracing a ray in a particular direction can be of use in both terms. As an example, if a ray is traced for the reflective component, the data obtained can be used for one of the diffuse rays. The only difference between the diffuse and reflective components is the weighting values. The weighing values for each component are summed and a single weight applied to the ray.

3.4.1. Distributed Ray Tracing

To improve the range of information obtained by each ray, parts of Equation 3.3 are implemented using a distributed ray tracing algorithm. This method, introduced in Chapter 2, redirects some of the secondary rays from their normal reflective or refractive directions. Applying this technique to Equation 3.3 can drastically reduce the number of secondary rays required per incident ray to adequately sample the secondary light sources around the surface. This section will examine the application of distributed ray tracing to the ray traced diffuse component of Equation 3.3.

Each pixel in the scene is oversampled at a minimum rate to obtain an initial scene sample. The pixel then can be viewed as being divided into an array that is a subpixels wide and b subpixels high. Rays are traced through the corners of a subpixel to reduce the sampling errors. It is assumed that the filter function applied to each pixel includes contributions from all incident rays traced within a pixel. The number of secondary diffuse rays generated from all the incident rays which pass through a single pixel is $N_{RD}(a+1)(b+1)$, where N_{RD} is the number of diffuse directions tested per intersection of a ray with an object. Each of these rays are directed into different areas of the image to obtain the greatest range of lighting contributions from all secondary light sources. This technique allows the value of N_{RD} to be small and still have a sufficient number of directions tested for the diffuse component.

3.5. Results

Equation 3.6a is the new lighting model and Equation 3.6b is Whitted's lighting model. They are presented here for reference.

$$I = K_D \sum_{i=1}^{N_{PL}} I_{L_i} \cos(\theta_i) + \sum_{i=1}^{N_{RD}} R_{D_i} W_{R_{D_i}} + \sum_{i=1}^{N_{RS}} R_{S_i} W_{R_{S_i}} + \sum_{i=1}^{N_{TD}} T_{D_i} W_{T_{D_i}} \quad 3.6a$$

$$I = K_D \sum_{i=1}^{N_{PL}} I_{L_i} \cos(\theta_i) + I_A + R_S K_R + T_S K_T \quad 3.6b$$

The image produced using the new lighting model clearly shows many diffuse lighting effects. All of the following images were obtained using the lighting models shown in Equations 3.6a and 3.6b. The implementations techniques presented in this chapter and in Chapter 2 were used to implement the models. All the images were rendered on a Vax 11/780 computer with a floating point accelerator. The images in Plates 3.1 and 3.2 were rendered at a resolution of 512 by 512. The images in Plates 3.3, 3.4, and 3.5 were rendered at a resolution of 512 by 384.

The image shown in Plate 3.1a is a simple empty room first introduced by Goral, Torrance, Greenburg, and Battaile [19] to show diffuse color bleeding effects. Whitted's lighting model was used to render the image in Plate 3.1a. The ray traced terms of Whitted's model are not used as there are no reflective or refractive surfaces. The ambient lighting factor is set to account for 30 percent of the diffuse component of each surface. The diffuse coefficient for the primary light sources is set to account for 70 percent of the total diffuse component. The light source is in the center of the image, just in front of the room. The diffuse lighting effects from the primary light source is accurately rendered using Whitted's lighting model. The back wall, which is perpendicular to the light source is the brightest. The four side walls show a smooth shading effect, being brighter in the front where the angle between the surface normal and the light source is smaller than in the back. The diffuse lighting effects missing from Plate 3.1a are the diffuse lighting contributions from the secondary light sources. These should appear as two main lighting effects. First, the side walls of the room should be slightly brighter in the back due to the light reflected from the back surface. Second, there should be some color bleeding from the walls where they join.

The image shown in Plate 3.1b was rendered using the new lighting model. The diffuse coefficient of the surface was set to 70 percent. The reflective and transmitted terms were not used as the scene contains no reflective or transparent surfaces. Only a single diffuse ray is traced for each incident ray ($N_D = 1$). By oversampling the image

at a rate of sixteen and using a distributed ray tracing algorithm, twenty five different diffuse directions were examined to determine the intensity of a each pixel. The image in Plate 3.1b is brighter in the back of the room due to the contributions from secondary light sources in the diffuse lighting calculation. An additional diffuse effect is the color bleeding on the floor from the walls. This bleeding effect is present at all corners of the room. These are the same effects obtained by Goral, Torrance, Greenburg, and Battaile using the radiosity method described in Section 2.2. The image in Plate 3.1b took twice the computing time as the image in Plate 3.1a.

A second example of the performance of the new lighting model is shown in Plates 3.2a and 3.2b. Figure 3.2 is a top view of the scene rendered in these plates.

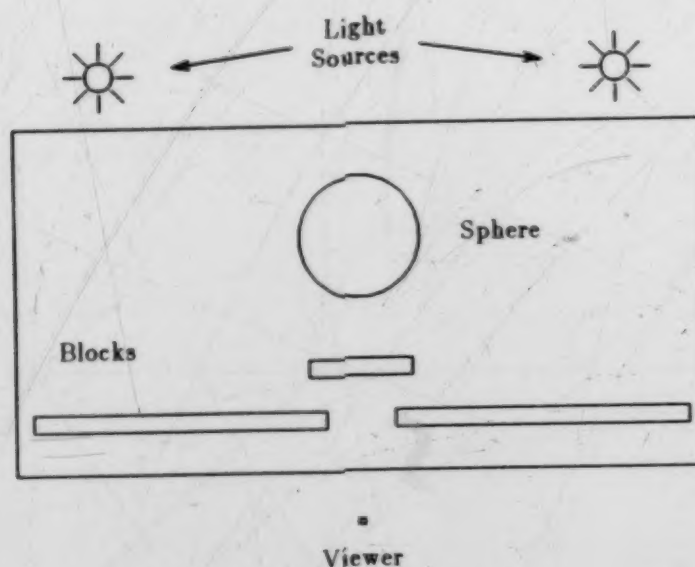


Figure 3.2 Diffuse Lighting on Blocks

The backs of the large blocks on the right and left of the scene are colored red and green respectively. The smaller center block is placed back slightly from the front blocks. A large reflective sphere is placed behind all the blocks so that the viewer can

see that the back of the blocks are colored. The blocks are illuminated from the back with the two light sources as shown. The fronts of the block receive no direct illumination. The image in Plate 3.2a was rendered using Whitted's lighting model. Each of the diffuse surfaces have an ambient coefficient of 30 percent and a diffuse coefficient for the primary light sources of 70 percent. The front of the blocks in Plate 3.2a have no direct illumination and the intensity of the surface is due completely to the constant ambient term. This gives the blocks a washed out appearance. The image in Plate 3.2b was rendered using the new lighting model. The diffuse coefficient is set to 70 percent. Only one diffuse ray is generated per incident ray and the image is over-sampled at a rate of sixteen. The image in Plate 3.2b has a much richer lighting effects. The front of the blocks are now brighter near the bottom due to the secondary light obtained from the floor. The center block shows bleeding effects from the back of the two end blocks. The right side of the center block is tinted red from the back of the right block, and the left side of the center block is tinted green from the back side of the left block.

3.5.1. Proposed Diffuse Test Image

The diffuse lighting effects shown in Plates 3.1 and 3.2 were obtained at the small cost of tracing a single extra ray per incident ray. The diffuse reflections between objects can quickly become much more complex than the previous examples showed. The scene used in the next three plates was designed to have a large range of diffuse lighting complexity. This image is used to study the effectiveness and limits of the new lighting model. Before discussing the images shown in these plates, the exact geometry of the scene is described.

The scene consists of a large room containing two objects, the viewer, and the light source. The room is 16 units wide, 6 units high, and 24 units deep. The walls, floors, and ceiling are perfect diffuse reflectors. The color of each of the walls, the

floor, and the ceiling is listed in Table 3.1.

Room			
Surface	Size	Color	Lighting
Floor	16x24	.8R .8B .8G	.7 Diffuse
Ceiling	16x24	.25R .25B .8G	.7 Diffuse
Left Wall	6x24	.8R .25B .25G	.7 Diffuse
Back Wall	6x16	.8R .8B .8G	.7 Diffuse
Right Wall	6x24	.25R .8B .25G	.7 Diffuse
Front Wall	6x16	.8R .8B .8G	.7 Diffuse
Box			
Surface	Size	Color	Lighting
Floor	1x4.6	.8R .8B .8G	.7 Diffuse
Ceiling	1x4.6	.25R .8B .25G	.7 Diffuse
Left Wall	2x4.6	.25R .25B .8G	.7 Diffuse
Back Wall	1x2	.8R .8B .8G	.7 Diffuse
Right Wall	2x4.6	.8R .8B .8G	.8 Ref .2 Diffuse
Ball			
Surface	Size	Color	Lighting
Ball	Dia 1.5	.8R .8B .8G	.7 Tran .3 Diffuse 1.8 Index
<i>Room - Middle of Back Left Corner is The Scene Origin</i>			
<i>Box - Rot -22 Deg About y (RHR), Center (14.74z, -1y, 3.25z)</i>			
<i>Ball - Center (14.5z, -1.5y, 9.5z)</i>			
<i>Light - Center (12z, -1y, 22z)</i>			
<i>Viewer - Center (8z, -1y, 23z), At (8z, -1y, 12z)</i>			
<i>Aspect Ratio - 4/3, Angle of View - 90 degrees</i>			

Table 3.1 Diffuse Test Data

The room contains two simple objects. The first is a glass sphere with a diameter of 1.5 units. The sphere is centered on the floor 14.5 units from the back wall and 9.5 units from the left wall. The sphere has an index of refraction of 1.8 and is 70 percent transparent and 30 percent diffuse. The second object is an open ended box. The box is 2 units high, 1 unit wide, and 4.6 units deep. The box is centered 14.75 units from the back wall and 3.25 units from the left wall. The box is rotated 22 degrees about its center exposing the open face and the right wall (-22 deg about the y axis using the right hand rule). All the walls of the box are perfect diffuse reflectors except the right

wall which is a mirror on both sides. The mirrored wall is 80 percent reflective and 20 percent diffuse. The light source is a white point source located 2 units from the floor, 12 units from the left wall and 22 units from the back wall. Finally, the viewer is located 8 units from the left wall, 23 units from the back wall, 2 units from the floor. The direction of view is the center of the room and down one unit. The aspect ratio is $4/3$ and the full angle of view is 90 degrees. These values are summarized in Table 3.1. The scene was designed such that the lighting effects are strongly interrelated and complex.

Plate 3.3a is this scene rendered using Whitted's lighting model. The sphere and the far right wall are visible in the reflection in the mirror on the side of the box. The box cast a shadow on the left wall. The interior of the box is obtains only a sliver of direct illumination from the light source. The view through the glass sphere is a distorted view of the room. The scene contains many errors resulting from not including global lighting effects in the diffuse lighting calculations. The first of these effects is the lack of color bleeding between joining surfaces. This effect should be most apparent in the open box and along the left wall of the room in the shadow of the box.

Another missing lighting effect occurs in the shadow of the sphere. Glass objects do not cast regular shadows and many researchers omit shadow from glass objects. If the diffuse lighting was correctly rendered, the light passing through the sphere would be focused into a spot whose size is dependent on the index of refraction of the glass, location of the sphere, and location of the surface. This shiny spot from the sphere should be visible in the reflection of the sphere in the mirror.

A similar effect should occur in the area in front of the mirror on the box. This area should be substantially brighter due to the extra light obtained from the mirror. This light is, in part, from the objects in the room, but also from the primary light source. Each point in front of the mirror which can see the light source in the mirror

will be about twice as bright as the areas which only obtain a single contribution from the light source. This diffuse lighting effect is similar to the effect of a projector on a screen. The light reflected from the mirror will fall on the sphere and the sphere should cast another shadow and focus point to the right.

Plates 3.3b, 3.4a, and 3.4b are images of this scene produced using the new lighting model. Plate 3.3b was rendered at an oversampling rate of one. This resulted in only four different diffuse directions being tested for each pixel. The diffuse lighting effects in this plate appear as multiple reflections and not as smooth diffuse lighting. This plate shows the general area and color of the diffuse lighting contribution from the secondary light sources. The areas around the sphere and the box all have some diffuse reflections appearing. There are further diffuse effects along the corners of the room. Even at the low sampling rate used for this image the diffuse effects in the back corners of the room look reasonable.

Plate 3.4a was rendered at an oversampling rate of four. This resulted in nine different diffuse directions being tested per pixel. The diffuse lighting effects no longer look like multiple reflections but the sampling errors are very pronounced. The diffuse lighting effects along the back wall look reasonable and the diffuse effects along the left wall of the room are smoothing out. The diffuse effect around the sphere and the box are still quite poor.

Plate 3.4b was rendered at an oversampling rate of sixteen. This results in twenty five different diffuse direction being tested per incident ray. The simple diffuse effects, such as the color bleeding in the corner of the room appear smoothly shaded, but many other diffuse effects still have sampling errors. Further increases in the sampling rate to reduce these sampling errors should be done by increasing the number of diffuse rays traced per incident ray. This will cause the extra rays to be generated where the sampling errors exist.

By changing the number of diffuse rays generated per incident ray from one to two, fifty different diffuse directions are tested per pixel. The image in Plate 3.4b required over 10 million rays to trace. The sampling errors should begin to disappear completely before 100 million rays are traced, but no such test image was rendered. One last interesting diffuse lighting effect which occurs in this set of images is found in the shadow of the box on the left wall and floor. The color bleeding effect on the floor between the wall and the box appears stronger further from the wall. This effect is due to the contribution of secondary light from the illuminated part of the wall above the shadow. The area of the floor away from the wall receives more of this light than the floor next to the shadowed wall.

The bright spot in the shadow of the glass sphere and the bright area in front of the mirror never really started to appear as the sampling rate was increased. There is some light on the floor in front of the mirror, but this is due to the diffuse component of the mirror and not due to the light reflected from the mirror. These effects are mainly due to a bright light source, such as the primary light source, illuminating the surface in more than one way. In the case of the surface in front of the mirror, the surface receives light directly from the light source and also as a reflection from the mirror. This second contribution was never realized because the light sources are modeled as points and the diffuse rays can never strike them. To simulate this effect a white sphere is placed around the point light source so that the diffuse rays have a bright object to intersect.

Plates 3.5a and 3.5b were rendered with this spherical light source. Plate 3.5a was rendered with an oversampling rate of four and Plate 3.5b was rendered with an oversampling rate of sixteen. Notice the spots which are appearing in the area in front of the mirror as expected. Also notice the spots appearing in other areas all around the scene. These other spots are due to a diffuse ray from a point on the surface hitting the light source, which is now represented with the sphere. The contribution of

the primary light source at these locations is being counted twice, once with Lambert's cosine law by the first term of Equation 3.6a and once by the ray traced diffuse lighting term. The cosine function could be eliminated so that there would be only one contribution, but an unreasonable number of diffuse rays would be required to obtain a smoothly shaded image (billions). Experiments with this technique using an oversampling rate of sixteen resulted in very dark images. The second approach is to divide the scene into primary and secondary light sources and insure each source is counted only once.

This is the motivation used to construct Equation 3.6a. The mirror in this scene is a bright light source which cannot be described as a collection of point light sources. This diffuse lighting effect from the mirror and the sphere cannot be included in the image by Equation 3.6a without degrading the equation to a massive sampling function. A third possible solution to this problem is to trace ray from multiple sources. One method would be to trace rays from the viewer and the light sources independently and merge the results. This approach is the topic of further research.

3.6. Summary

The lighting model presented in this chapter models the three main lighting components of a surface, in a simple and uniform manner. Several images were presented which showed that while the model does include many diffuse lighting effects it is not the final answer to this problem. This model renders images more accurately than was possible with previous models.

Techniques were introduced to reduce the number of rays traced by the rendering system to obtain the lighting information required by this model. The effectiveness of the model is related to the time required to produce the images. Further and more powerful methods of reducing the rendering time of a ray tracing system are presented in the following chapter.

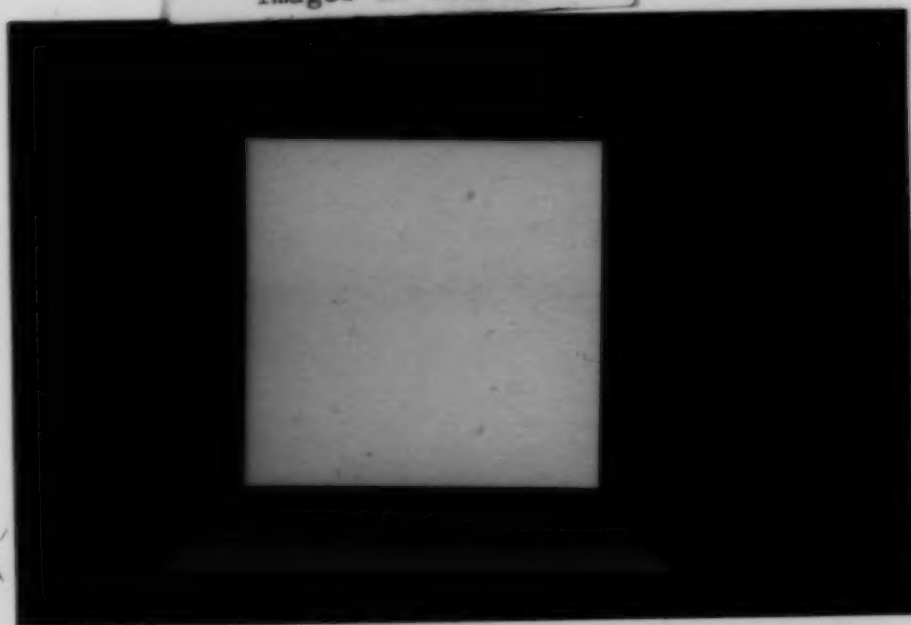


Plate 3.1a



Plate 3.1b

Plate 3.1 Diffuse Lighting Plate 1



Plate 3.2a

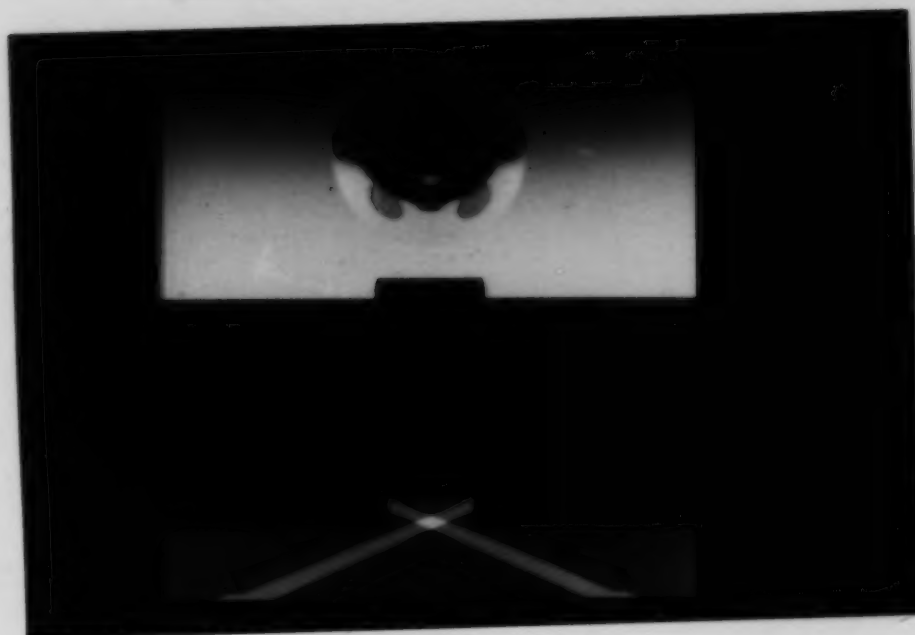


Plate 3.2b

Plate 3.2 Diffuse Lighting Plate 2

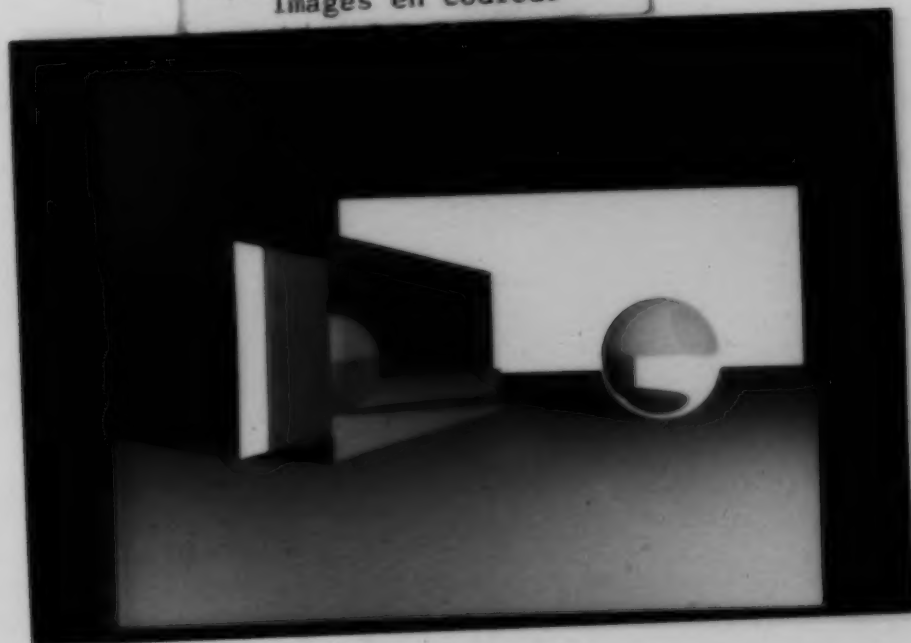


Plate 3.3a

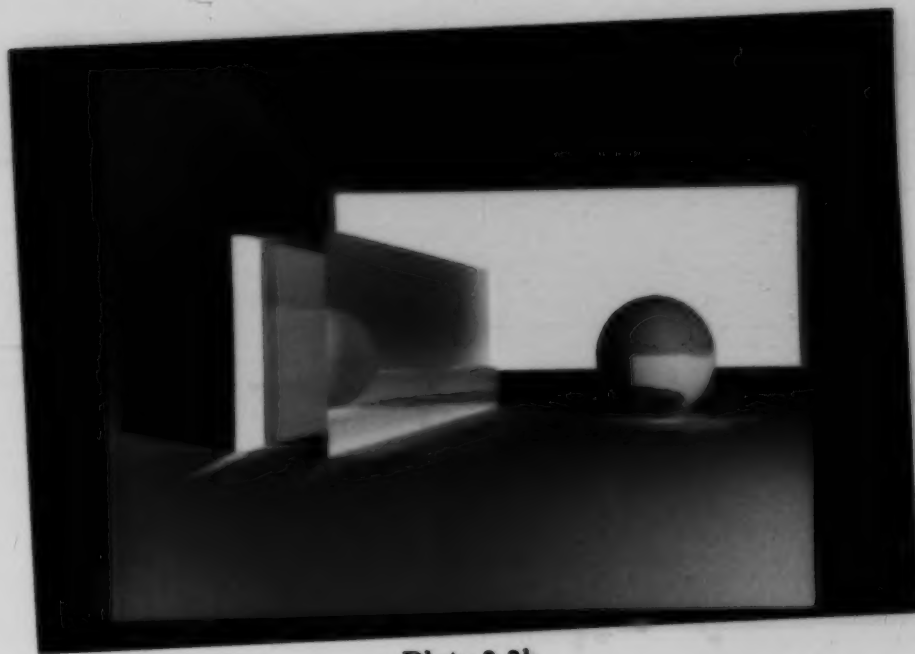


Plate 3.3b

Plate 3.3 Diffuse Lighting Plate 3

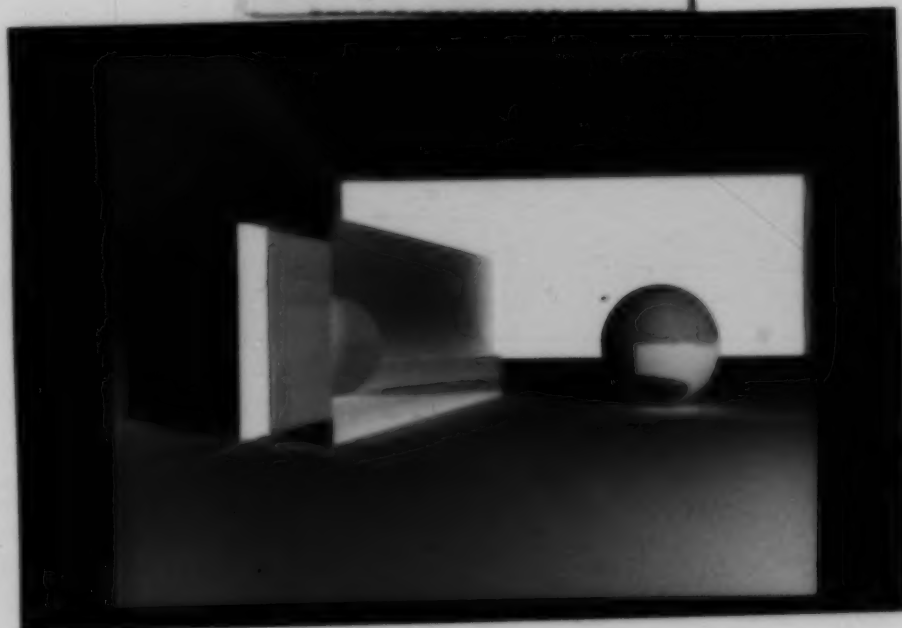


Plate 3.4a

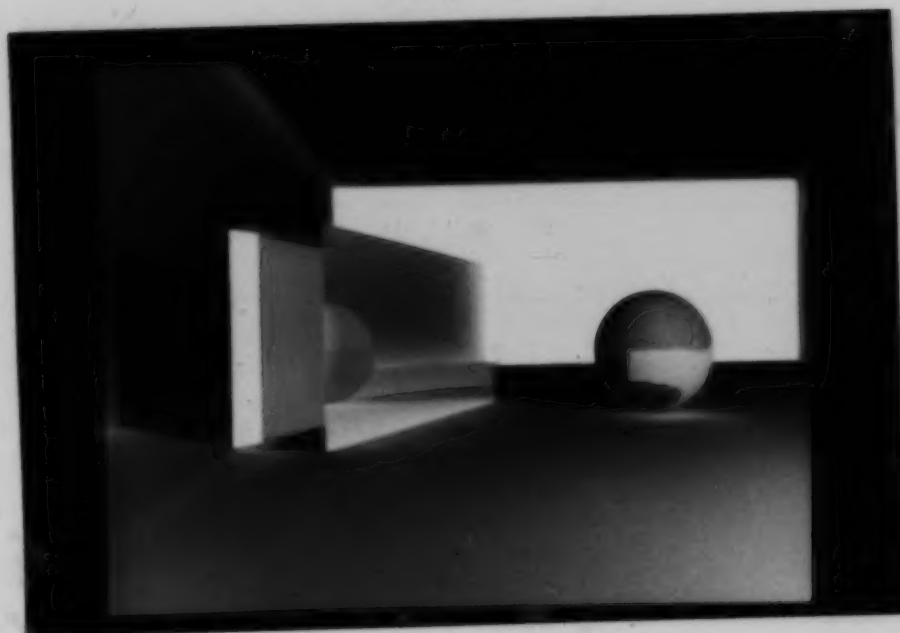


Plate 3.4b

Plate 3.4 Diffuse Lighting Plate 4

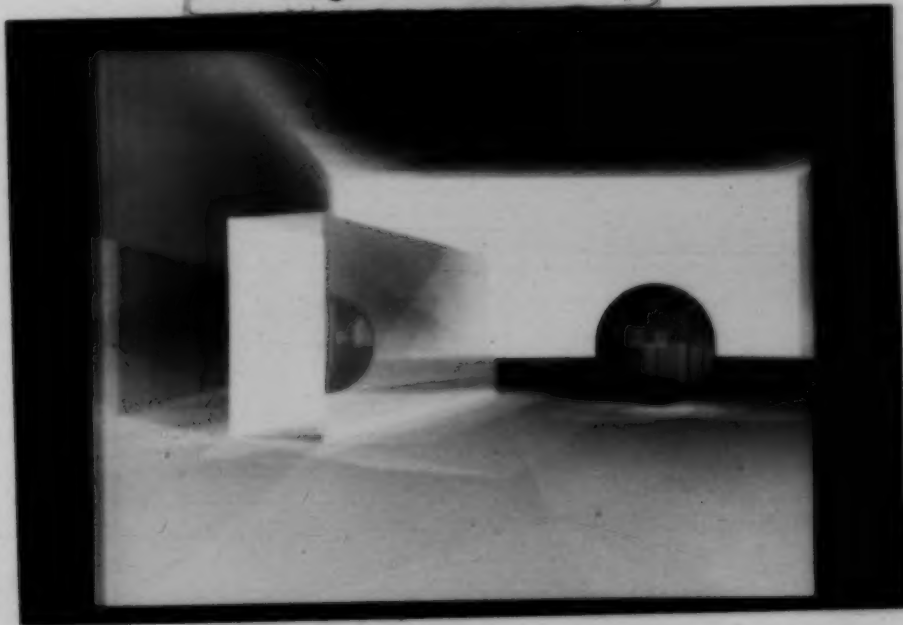


Plate 3.5a

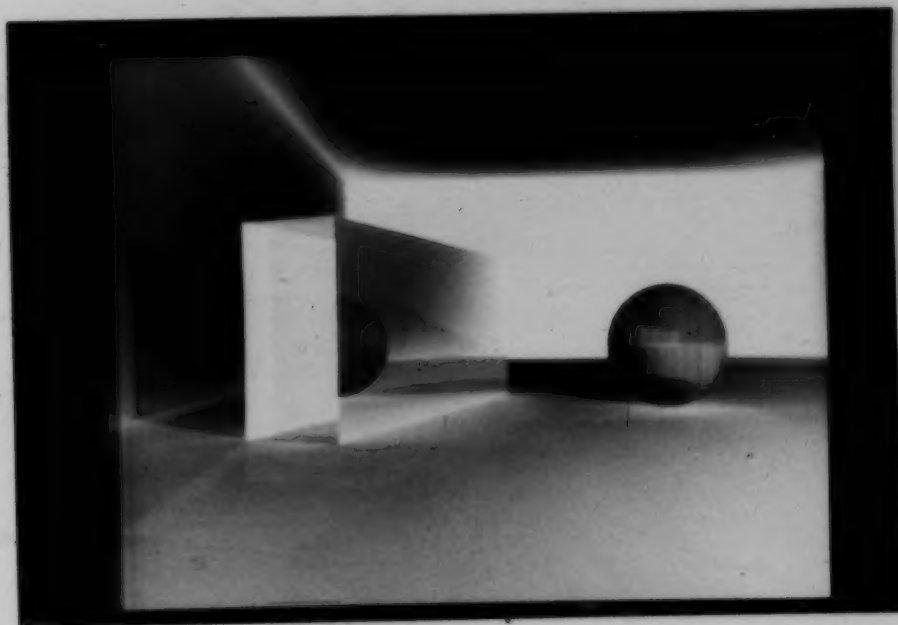


Plate 3.5b

Plate 3.5 Diffuse Lighting Plate 3

Chapter 4

Data Hierarchy

4.1. Introduction

Ray tracing is an attractive solution to the hidden surface problem as it naturally accounts for many image features such as global lighting effects, shadows, and transparent objects. Ray tracing algorithms must consider each data primitive in the scene to find the visible primitive for each ray. This process can be excessively slow due to the large number of rays required to render an image and the large number of data primitives used to define a scene.

The key factors used to improve the efficiency of ray tracing algorithms is to presort the image before rendering and to take advantage of both image and frame to frame coherence. The level of efficiency obtained is dependent on the type of preprocessing performed on the data, the type of traversal algorithm used during rendering, and the number and type of frames required. The data hierarchies and traversal algorithms presented in Section 2.6 possess many of the properties necessary for this purpose.

A robust data hierarchy and traversal algorithm is presented in this chapter as an alternative to existing proposals. This hierarchy is more flexible, smaller, and shows improved traversal speed over existing data structures on most images, especially the class of images that contain large empty areas.

A parallel ray tracing algorithm is defined using the new hierarchy. A multiprocessor ray tracing system is introduced that is based on the parallel ray tracing algorithm. This ray based system offers greater rendering efficiency over existing parallel ray tracing systems by improving processor usage and minimizing interprocessor communications.

4.2. Object/Volume Hierarchy

The standard object based and volume based hierarchies on which this work is based were presented in Section 2.6. These structures used bounding volumes and data ordering to reduce the time required to find the closest primitive which intersects a ray. A new hierarchy is constructed which combines the strengths of object and volume based hierarchies into a single structure. This new hierarchy is called an object/volume hierarchy to reflect this combination.

The most visible feature of the object/volume hierarchy is the use of an object based hierarchy at the upper levels of the structure and a volume based hierarchy at the lower levels. Figure 4.1 shows the general structure of the hierarchy. The object based section of the object/volume hierarchy is similar to the object hierarchy discussed in Section 2.6 with some changes to the scope of the hierarchy and its traversal. The volume based sections of the object/volume hierarchy is substantially different than the volume hierarchies presented in Section 2.6 and is discussed in greater detail.

The construction of the object/volume hierarchy is presented first. This is followed by the algorithm used to traverse the hierarchy.

4.2.1. Construction

The construction of the object and volume based sections of the object/volume hierarchy are presented independently. The construction of the object section is presented first.

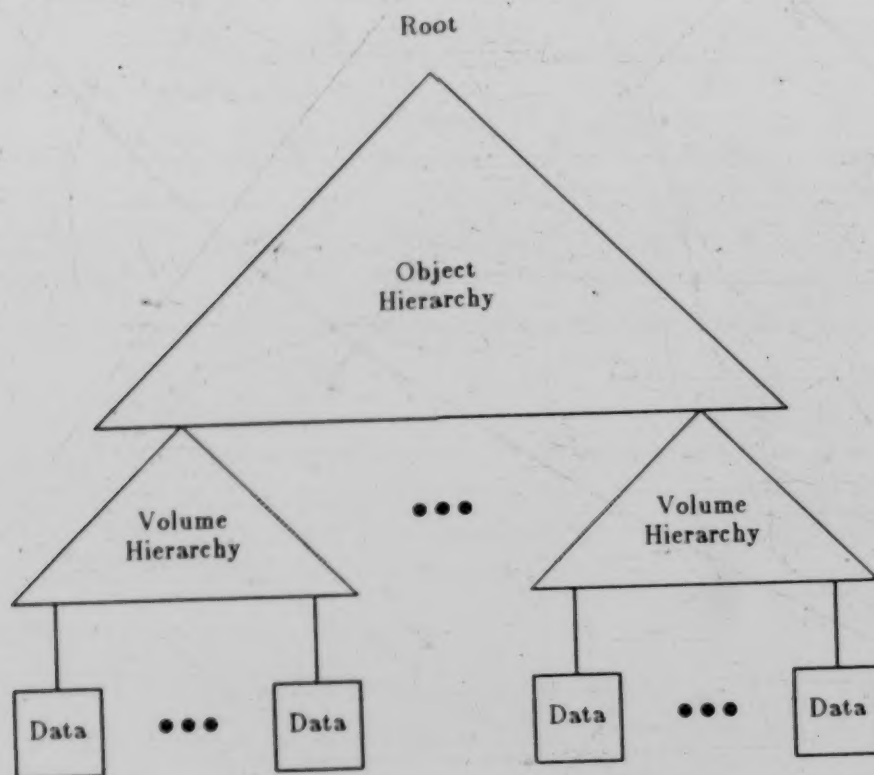


Figure 4.1 Design of the Hierarchy

4.2.1.1. Object Hierarchy

The initial construction of the object hierarchy is done manually. This section presents the construction of the object hierarchy from the bottom up. The simplest objects in the object hierarchy are collections of primitives which remain geometrically fixed relative to each other. Such a collection of primitives is referred to as a part. Complex objects are constructed as collections of these simple parts. The classification of a group of primitives as a part and the collection of different parts into objects is dependent on the scene being rendered and the requirements of the animation sequence. This procedure is demonstrated with a simple example.

Consider the collection of primitives used to describe a car. The designer of the hierarchy plans to leave the car parked during the entire animation sequence. The designer knows that the car will be placed on some street and possibly its size changed to match the physical description of the street, but the car doors will not be opened, the wheels turned, or any other part of the car moved relative to itself. The entire car is then considered a single part and stored as a leaf node of the object hierarchy. The next higher level object could be a road, with a street and several cars as children. In another rendering situation the designer knows that the wheels of the car are turned and the doors are opened and closed at some point during the animation sequence. The designer then makes each door and wheel a separate part. The description of the car is a higher level object consisting of the wheels, doors, and the body of the car as its children. The car is then included in the larger hierarchical description of the scene.

The different parts of the car are moved using the transformation matrixes stored at each object node as discussed in Section 2.6 for conventional object hierarchies. Simpler objects are collected into larger objects until all the objects in the scene are contained under a single object node known as the root. The resulting object hierarchy may contain large leaf nodes, such as the car. If this node is visited during rendering each of the primitives must be tested for intersection with the ray, requiring considerable computation. To reduce this intersection time, the leaf nodes of the object hierarchy are subdivided further into the volume based oct-tree structure described next. The relationship between the object and volume based sections of the hierarchy is shown in Figure 4.1.

Object hierarchies were shown to be robust structures in Section 2.6. This property is preserved in the object hierarchy described in this section. Object hierarchies obtained their robustness from the transformation matrices stored at each node and the independence of one node from the next. Between frames of an animation sequence

the transformation matrices at the objects nodes are modified to move items, and the links between object nodes created or deleted to add or remove items. The construction procedure outlined in this section requires object nodes to be used for all parts that have some degree of independent motion. This insures that transformation matrices and object node links are available for updating, creating or deleting at the necessary locations during the animation sequence.

The object hierarchy contains other structures, such as bounding volumes, which are used during rendering. These items are introduced and explained as required during the discussing of the traversal algorithm.

4.2.1.2. Volume Hierarchy

The construction of the volume hierarchy is presented from the top down. Each of the leaf nodes of the object hierarchy are divided into individual volume hierarchies. This section will examine the construction of the volume hierarchy from one of these leaf nodes. An actual tree containing interior and leaf nodes is created by this process.

The part stored as a leaf node of the object hierarchy is subdivided into an volume based oct-tree structure. The first step in the construction is to examine the part to determine the center of the data and the smallest bounding box which encloses the entire part. A decision then is made to determine if this volume requires subdivision. This decision is based on the size of the volume, the number of primitives in the volume, or a combination of both. The intent of this decision is to reduce the number of primitives in a single volume to a manageable level. The optimal size of a volume is a function of the amount of processing and storage overhead required for the hierarchical structure created from this volume, relative to the time required to simply intersect the primitives in the volume during rendering.

If the volume requires no further subdivision then the data are left as a leaf node to the volume hierarchy. Otherwise the data are divided into eight equal subvolumes

about the center of the initial volume. An interior node is created and each subvolume is connected as a child to this interior node. Each child contains only the data which falls in its assigned subvolume. For each of the subvolumes, the data are examined again to find the center of the data and the minimum bounding box in an analogous manner to the creation of the initial volume. A decision is made to determine if each subvolume requires subdivision and the process is recursively repeated. A tree is created by this process. The root of the tree is connected as a leaf node to the object hierarchy. The leaf nodes of the volume hierarchy contain the data. Figure 4.2 shows a sample object/volume hierarchy.

The clipping and traversal calculations are simplified if the bounding boxes are aligned with the global coordinate axis. This can be done by requiring the boxes to be aligned with the coordinate axis when they are created or by including a transformation matrix with the box which will realign it with the coordinate axis when required [31]. The advantage of the second approach is that a smaller box may be found, thus potentiality improving the rendering speed. This must be weighed against the time required to transform the data, ray, or box at different points in the rendering process. Complex animation effects, such as motion blur [12], can require numerous transformations to be done. The bounding boxes used in this thesis are all aligned with the coordinate axis but the best method for a given application is dependent on the available hardware and animation effects required.

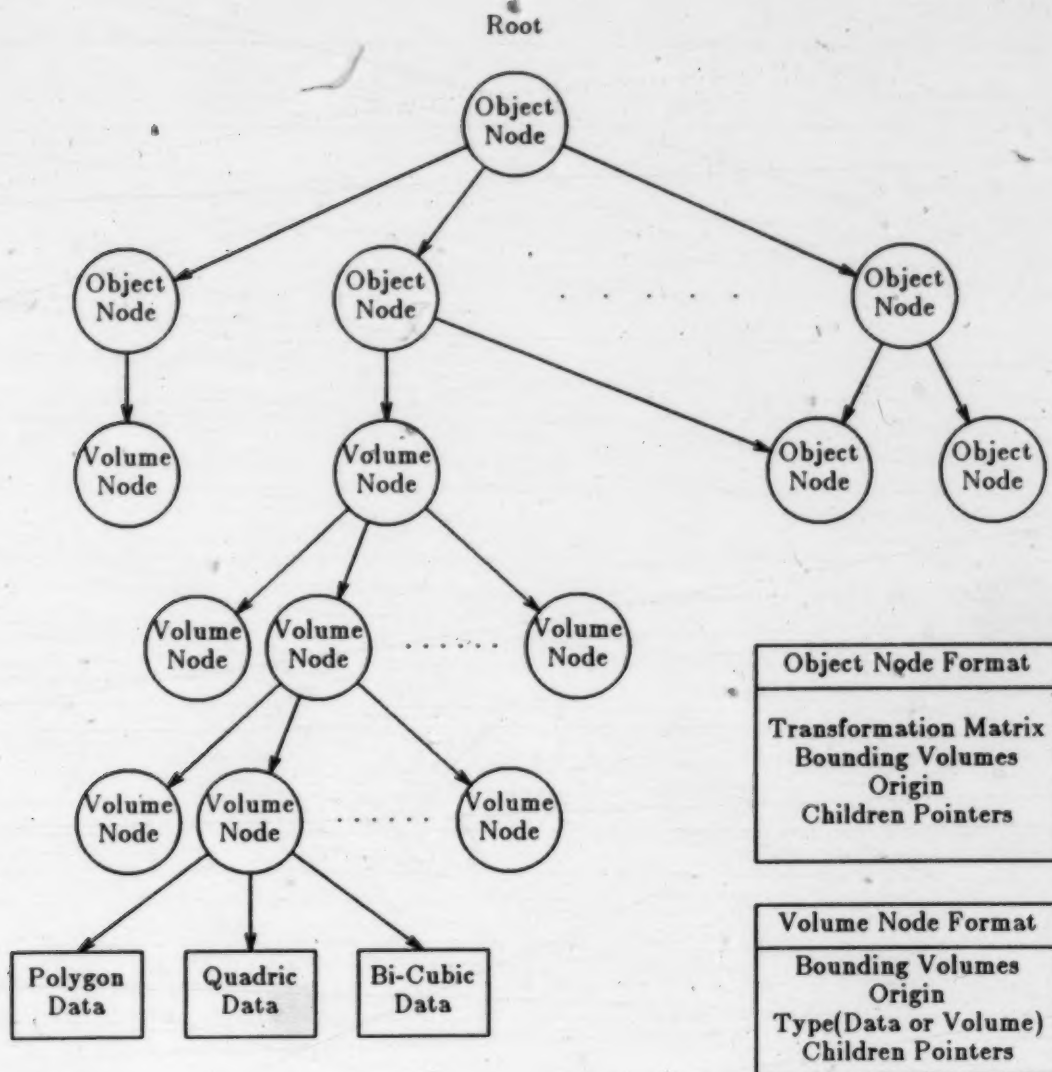


Figure 4.2 Object/Volume Hierarchy

The relationship between the children and the subvolumes they represent must be preserved so that the structure can be properly traversed during the rendering process. The center of the data at a node is used as the origin of the local coordinate system for that volume node. This system is aligned with the global coordinate axis. Table 4.1 shows which child of a given node contains what subvolumes of the divided volume

relative to the local coordinate system.

Child	0	1	2	3	4	5	6	7
x - Origin	+	+	+	+	-	-	-	-
y - Origin	+	+	-	-	+	+	-	-
z - Origin	+	-	+	-	+	-	+	-

Table 4.1 Volume-Child Relationship

Figure 4.3a shows a two dimensional view of a part that has been subdivided using the regular volume hierarchy [40] or the adaptive volume hierarchy [18] introduced in Section 2.6. Figure 4.3b shows the same part subdivided using the object/volume hierarchy.

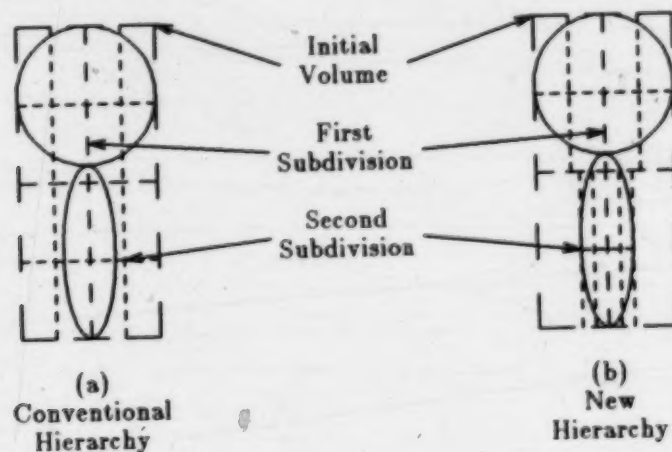


Figure 4.3 Data Subdivision

The initial volume is represented by the root of all three hierarchies. This volume is subdivided about the center creating four equal subvolumes (eight for the three dimensional case). The conventional volume hierarchies then divides each of these subvolumes into four identical volumes. This results in sixteen equal size volumes. The new hierarchy re-examines the data in each subvolume and defines a smaller volume if possible. This smaller volume is then divided into four equal parts. This

results in sixteen unequal size subvolumes. The process continues using the subdivision criteria for each hierarchy until the data are completely subdivided.

One of the key features of the volume section of the object/volume hierarchy is that each volume node is subdivided about the center of the data at that node. This approach reduces the amount of empty area explicitly represented in the hierarchy and tends to evenly divide the data between each child. This can be seen in Figure 4.3. The empty areas on each side of the ellipse are included in the second subdivision in the conventional volume hierarchies but eliminated in the construction of the volume section of the object/volume hierarchy.

The size of the object/volume hierarchy will be the nearly the same size or smaller than the regular and adaptive volume hierarchies if the scene has open spaces and compact objects. If the scene is very complex with little open space the overhead incurred by the object/volume hierarchy may cause the hierarchy to become larger than the conventional volume hierarchies.

The three different construction techniques used for each volume hierarchy requires three different traversal algorithms. The traversal of the regular volume hierarchy, the adaptive volume hierarchy, and the object hierarchy was introduced in Section 2.6. The traversal of the object/volume hierarchy is presented next.

4.2.2. Traversal

The main benefits of the object/volume hierarchy are exploited by the traversal algorithm used in the rendering process. The hierarchy is traversed to find the data primitive which intersects the ray and is closest to the ray's origin. This operation is examined in detail in this section.

To reduce the time required to traverse the hierarchy, large portions must be eliminated from consideration early in the search. This is partially done with the use of bounding volumes at each node of the hierarchy. If a ray does not intersect the

bounding volumes at any node then it cannot possibly intersect any of the data primitives below this node in the hierarchy. The construction and types of bounding volumes was introduced in Section 2.6 and is examined further.

Traversal begins by intersecting the ray with the bounding volumes at the root of the hierarchy which is always an object node. If the ray fails to intersect any of these bounding volumes the ray will not intersect any of the data primitives below this node and the traversal process terminates and returns a fail. The intersection test with the bounding volumes at this level need only be pass-fail as the exact point of intersection is not necessary.

If the ray intersects all the bounding volumes at the root node then each of the children of that node must be tested. The bounding volumes at each child are tested for intersection with the ray and those failing the test ignored. There usually will be only a few children which pass the bounding volume test. The objects represented by these passing nodes are found one in front of the other along the path of the ray. It is always assumed the ray will intersect one of these objects first. The points of intersection of the ray with a bounding volume of the passing object nodes are found and the nodes sorted by distance from the origin of the ray. The closest object node is expanded first. The result of expanding this node can be a fail, indicating the ray did not intersect any of the data below this node, or a pass and the exact point of intersection with an object returned. If a fail is returned the next node in the sorted list is expanded. If a pass is returned this intersection point is compared with the distance the next node in the sorted list is from the origin of the ray. If the intersection point is closer than the next node, the node need not be expanded because any primitive which intersects the ray inside the bounding volume cannot be closer than the current point. If there are any remaining volumes on the sorted list they are also discarded for the same reasons. The only time more than one object node will be expanded at any level is if the objects are on top of each other. While this situation is quite possible, it is

unlikely. This algorithm is used until the volume hierarchy is reached. On the average only one node is expanded at any level and thus only one volume hierarchy will be traversed.

The volumes which contain data in a volume hierarchy must be considered in the same order that a ray encounters them while traveling through the scene. Ullner [40] solved this problem by using a regular volume hierarchy where every volume is of identical size. The first and next volumes the ray enters are directly determined and visited. Glassner [18] solved this problem by using a hash table to find a volume given its name. The name is found from any point in the volume by traversing a tree which contains the information on how the hierarchy was created. Glassner finds the next node by finding the point the ray leaves a volume and adding a small increment which is half the size of the smallest volume. The new point is guaranteed to be in the next volume. The name is found and the volume located. The algorithm presented for the object/volume hierarchy will consider each of these nodes in the same order the ray encounters them, but will actually visit very few of the nodes.

The nodes of the volume hierarchy contain bounding volumes identical to those used in the object hierarchy. The ray is tested against the bounding volumes at the root of the volume hierarchy. If the ray passes the bounding volume test one or more of the children will be expanded. As mentioned earlier, the children must be expanded in such a manner that the data nodes stored as leaf nodes of the hierarchy are considered in the same order as the ray visits them. To determine which child node to expand, first a comparison is made between the origin of this node and the current ray. Based on this comparison the child specified by Table 4.1 is visited. If there are no intersecting primitives found in the hierarchy below this child, the next subvolume the ray enters is determined. The ray will enter a new subvolume when it crosses any of the planes of the local coordinate system. As there are three such planes, up to three more subvolumes could be visited. Determining the next subvolume to visit involves

intersecting each plane with the ray and using the results to specify the next child. The intersection calculation is quite simple if the ray has been normalized to the local coordinates of the node. This calculation is further simplified because the planes are aligned with the local coordinate axis. The following equations are used to intersect the ray with each plane.

$$t_x = \frac{P_{1x}}{P_{2x} - P_{1x}} \quad t_y = \frac{P_{1y}}{P_{2y} - P_{1y}} \quad t_z = \frac{P_{1z}}{P_{2z} - P_{1z}}$$

where,

t_x, t_y, t_z = The parametric value along the ray.

P_{1x}, P_{1y}, P_{1z} = The origin of the ray.

P_{2x}, P_{2y}, P_{2z} = A point along the ray.

The values of t are compared to find which plane, if any, the ray crosses first. If the denominator of any of these equations is zero then the ray is parallel to the plane and t is undefined. If the value of t is undefined or negative the ray does not cross the appropriate plane. The next subvolume entered is dependent on the current subvolume and the order in which the planes are crossed. This calculation can be performed by adding an index to the current volume and taking modulo 8 of the result. Table 4.2 contains the indexes to be added to the starting subvolume to determine the next volumes to be visited. For example, if the initial subvolume to visit is child 3 and the relationship between the values of t is $t_x < t_y < t_z$, then line 11 of Table 4.2 is used. If no intersecting primitive is found by expanding the first child, the second child is determined by adding 4 to the number of the first child to obtain the child numbered 7. If no intersecting primitive is found by expanding this child, the third child to expand is obtained by adding 2 to the number 7 obtain the number 9 and taking modulo 8 for the child numbered 1. If no intersecting primitive is found from the expansion of this node, the last volume node to expanded is obtained by adding 1 to

the last child number of 1 to obtain the child numbered 2.

t_x	t_y	t_z	Relation	Second	Third	Fourth
undefined	undefined	undefined	na	na	na	na
undefined	undefined	≥ 0	na	1	na	na
undefined	≥ 0	undefined	na	2	na	na
≥ 0	undefined	undefined	na	4	na	na
undefined	≥ 0	≥ 0	$t_y < t_x$	2	1	na
undefined	≥ 0	≥ 0	$t_x < t_y$	1	2	na
≥ 0	undefined	≥ 0	$t_x < t_y$	4	1	na
≥ 0	undefined	≥ 0	$t_x < t_z$	1	4	na
≥ 0	≥ 0	undefined	$t_x < t_y$	4	2	na
≥ 0	≥ 0	undefined	$t_y < t_x$	2	4	na
≥ 0	≥ 0	≥ 0	$t_x < t_y < t_z$	4	2	1
≥ 0	≥ 0	≥ 0	$t_x < t_z < t_y$	4	1	2
≥ 0	≥ 0	≥ 0	$t_y < t_x < t_z$	2	4	1
≥ 0	≥ 0	≥ 0	$t_y < t_z < t_x$	2	1	4
≥ 0	≥ 0	≥ 0	$t_z < t_x < t_y$	1	4	2
≥ 0	≥ 0	≥ 0	$t_z < t_y < t_x$	1	2	4

Table 4.2 Next Child Index

The algorithms used to traverse the object/volume hierarchy are summarized in Figure 4.4. The routine *Traverse* is used to traverse the entire hierarchy for each ray. The routine *Object_Traverse* is used to traverse the object hierarchy and the routine *Volume_Traverse* is used to traverse the volume hierarchy.

The algorithms used to create the object/volume hierarchy are summarized in Figure 4.5. The initial data is read by the routine *Build_Hierarchy* and the object section of the hierarchy is constructed. The data is then subdivided recursively by the routine *Build_Volume* till the termination criteria is satisfied.

```

Traverse() {
    Add, Delete, and Update Object Hierarchy;
    For( Each Ray in Image ) {
        Current_Prim = NULL; Object_Traverse(Root,Ray);
    } }

    Object_Traverse(Obj,Ray) {
        If( Ray Strikes Bounding Sphere ) {
            If( Children are Object Nodes ) {
                Intersect Ray With Bounding Sphere for Each Child;
                Build Sorted List of Children By Distance;
                While( There is Another Child Closer then Current_Prim )
                    Object_Traverse(Obj,Ray);
            } Else {
                Volume_Traverse(Obj,Ray);
            } } }

        Volume_Traverse(Obj,Ray) {
            If( Ray Strikes Bounding Sphere ) {
                If( Children are Volume Nodes ) {
                    While( No Primitive Has Been Found Which Intersects The Ray
                        In a Child Node ) {
                        Get Next Volume The Ray Will Enter;
                        Volume_Traverse(Next_Volume,Ray);
                    } Else {
                        Check All the Data Find The Closest Primitive
                    } } } }
        } } } }

```

Figure 4.4 Traversal Code

```

Build_Hierarchy() {
    ° Create Root Object Node;
    For( Each Object Input ) {
        Make Object Child of Root;
        Set Max_Primitives;
        For( Each Part in Object ) {
            Build_Volume(Part);
            Make Volume Child of Part;
        }
    }
    Interactively Arrange Object Hierarchy;
}

Build_Volume(Volume) {
    If( The Number of Primitives in Volume < Max_Primitives ) {
        Clip Data About Center into Eight Sub-Volumes
        For( Each Sub-Volume ) {
            Build_Volume(Sub-Volume);
            Make Sub-Volume Child of Original Volume;
        }
    }
}

```

Figure 4.5 Construction Code

Due to the use of bounding volumes at each node of the hierarchy, very few leaf nodes which contain data will ever be visited. If the ray misses all of the objects in the scene, no data primitives may be tested at all and few interior nodes will be examined. If the ray does strike an object, chances are good that only the nodes on a direct path from the root to the node containing the data primitives for this object will be examined. The envisioned use of this structure is that the construction of objects into a volume hierarchy is done only once when the object is originally created. For a given

animation sequence, the designer will build an object hierarchy using these objects from a library.

The advantages of this structure over existing proposals is analysed in detail in the following section.

4.3. Analysis

4.3.1. Introduction

The efficiency of the object/volume hierarchy and traversal algorithm is dependent on the applications to which they are being applied. This analysis will assume that the designer is producing several images of a particular scene from different viewing positions or will add, delete, or move objects between several frames. This is the standard procedure in animation systems but is also useful in single frame rendering as it allows the designer to rearrange the image until the desired effect is obtained.

The object/volume hierarchy presented in this chapter is compared to a standard object hierarchy [41], a regular volume hierarchy [40], and an adaptive volume hierarchy [18]. The two criteria used to compare these hierarchies are the size of the hierarchy and the total rendering time.

4.3.2. Hierarchy Size

The size of a hierarchy is defined by the number and type of nodes and the number of bytes used to store these nodes. By their definition, each hierarchy is constructed in such a way that the final configuration of the hierarchy is dependent on the scene being rendered. Without a general method of classifying and evaluating images based on their geometric complexity an analytical comparison of these hierarchies is impossible. To substantiate the following analysis three test images were constructed. These images are shown in Plate 4.1 and Plate 4.2.

COLOURED PICTURES
Images en couleur

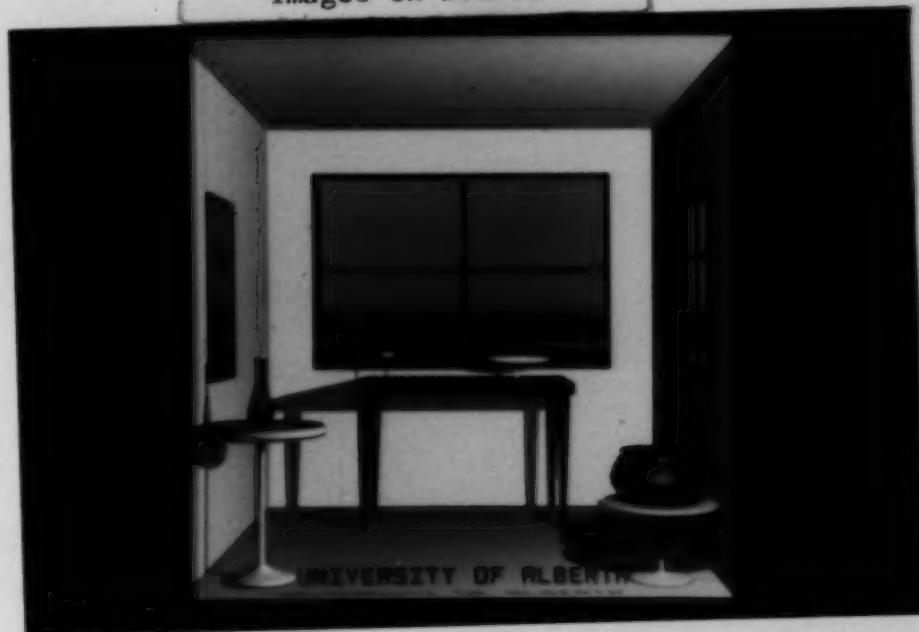


Plate 4.1a - Beach House

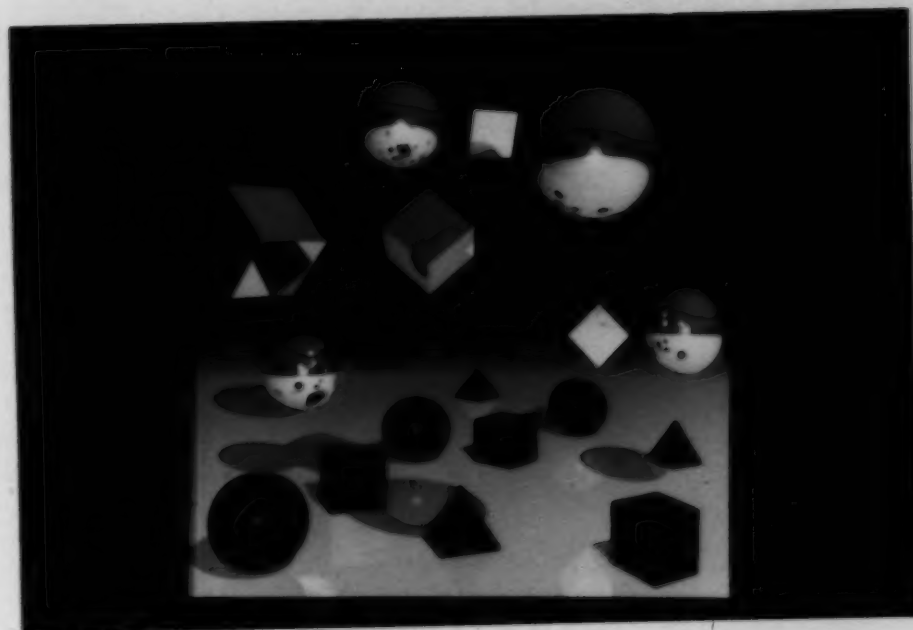


Plate 4.1b - Primitives

Plate 4.1 Test Images - Plate 1

COLOURED PICTURES
Images en couleur



Plant

Plate 4.2 Test Images - Plate 2

The images were selected to be representative of those images produced by graphics researchers and to have varying degrees of geometric complexity. The image shown in Plate 4.1a is a room with several small complex objects at various locations. The majority of the room is open and quite simple. The image shown in Plate 4.1b is a simple scene with many evenly distributed objects of very low geometric complexity. The final test image, shown in Plate 4.2 is a reasonably complex scene, with large complex objects in the center areas.

In the actual implementation of the four hierarchies, the image data is not stored at the leaf nodes of the volume hierarchy but as simple objects. These simple objects are linked together into a list of objects. The leaf nodes of the volume hierarchies contain pointers to the data in this list, whereas the leaf nodes of the object hierarchy point to one or more of these simple objects. The size and type of each of these simple objects for the three test images is given in Table 4.3.

Test Image 1 - Beach House			Test Image 2 - Primitives			
Name	Size	Color	Name	Number	Size	Color
Room	14P	white	Floor	1	1P	white
Sky	1P	blue	Sky	1	1Q	blue
Grass	1P	green	Spheres	7	1Q	various
Center Table	70P	brown	Boxes	7P	6	various
Dish	360P	white	Pyramid	3	5P	various
Sand	2P	tan	Total	66		
Water	3P	blue	Test Image 3 - Plant			
Name	Size	Color	Name	Number	Size	Color
Left Table	180P	white	Leaf	8	294P	green
Right Table	180P	white	Pot	1	280P	brown
Vase	80P	red	Dirt	1	36P	black
Glass	360P	gold	Stem	1	36P	green
Bowl	260P	blue	Corner	1	4P	white
Frames	80P	brown	Total	2708		
Title	67P	green	<i>P - Polygons, Q - Quadrics</i>			
Total	1658					

Table 4.3 Test Data Distribution

The efficiency of the intersection, shading and mapping calculations is improved by storing precalculated values with the data primitives. These include plane equations for the polygons and first derivatives for the quadric surfaces. Table 4.4 contains the size of the data primitives for polygons and quadric surfaces.

The size of a polygon is dependent on the number of edges and the shading method used. For each polygon the plane equation (4 floats), center (3 floats), and center normal (3 floats) is precalculated and saved. For each node of the polygon the coordinates (3 floats), normal (3 floats), and edge information (4 floats) is precalculated and saved. The size of a quadric surface depends on the number of planes used to delimit it. The quadric surface itself requires ten coefficients (10 floats) for its description and four coefficients (4 floats) for the first derivative (normal). Each plane requires the four coefficients for the plane equation (4 floats).

Linked Object List for Test Images		
<i>Image</i>	<i>Objects</i>	<i>Total Size</i>
Image 1	14	208,436 Bytes
Image 2	19	8,884 Bytes
Image 3	12	286,128 Bytes

Node Sizes		
<i>Hierarchy</i>	<i>Type</i>	<i>Size</i>
Object	Interior	88 Bytes
Regular Volume	Leaf	4 Bytes
Adaptive	Interior/Leaf	24 Bytes
Object/Volume	Object Interior	88 Bytes
Object/Volume	Volume Interior	24 Bytes
Object/Volume	Volume Leaf	20 Bytes

Primitive Storage Size		
<i>Primitive Type</i>	<i>Space</i>	<i>Comment</i>
Polygon	$40N+40$, N is # of nodes	200 Bytes for 4 edges
Quadric	$16N+56$, N is # of delimiting planes	152 Bytes for 6 planes

Table 4.4 Data and Node Sizes

While object hierarchies are often considered graphs, it is inefficient to implement them in this form unless special hardware is available. During rendering a composite transformation matrix is required for each node. If the object hierarchy is a graph there are many paths to a single node and hence many possible composite matrices. To keep track of the composite transformation matrix in a graph structure, two 4 by 4 matrices are required to be multiplied each time an object node is visited by a ray. The matrix then is applied to the data or its inverse applied to the ray. By restricting the hierarchy to a tree, these matrices can be determined at the start of the rendering process. This requires some of data to be duplicated but is well worth the price. The size of each primitive and the space required to store each of the images is listed in Table 4.4. The values in this table are slightly smaller than expected due to the sharing of nodes and edges in polygonal meshes. The storage required for the image data is common to each of the four hierarchies.

The space required to store each type of node is examined now. This section will assume that spheres are the only bounding volume used. The space required to store a bounding sphere simply requires the center (3 floats) and the radius (1 float) of the sphere to be stored. The space required to store an object node is the space for the bounding sphere, the space for one local transformation matrix (16 floats), a pointer to the first child (1 long) and a pointer to a sibling (1 long). By restricting the object hierarchy to a tree, an object node simply points to a linked list of children. The sibling pointer is used to link the children.

The storage required for the volume node in a volume hierarchy varies with the different types of volume hierarchies. All the leaf volume nodes contain a list of pointers to each of the data primitives in the list of simple objects which have some part in the leaf node's assigned volume. The regular hierarchy contains leaf nodes with these pointers plus an array of pointers to each of the leaf nodes. The adaptive hierarchy uses a hash table to find the leaf node which contains the required data. The recommended size of the hash table is 1000 entries. An auxiliary tree is used to save the data on how the tree was constructed and contains both internal and leaf nodes. Each node consist of a size (1 long), center (3 floats), a pointer to a child, a sibling pointer, and a flag indicating if the node was subdivided. The interior volume node of the object/volume hierarchy contains storage for a bounding sphere, a interior/leaf flag, a pointer to a child, and a sibling pointer. The leaf nodes of the object/volume hierarchy contain a bounding volume and a pointer to a list of data pointers. The space required for each of these nodes is summarized in Table 4.4.

Data structures were built for each of the test images using the four hierarchies described. The volume based hierarchies were built with ten, fifteen, and twenty five primitives maximum per volume as the subdivision criteria. The number and type of each node is listed in Table 4.5. The total values do not include the space for the linked list of simple objects.

Object Hierarchy			
<i>Image</i>	<i>Obj Nodes</i>	<i>Obj Pnts</i>	<i>Size</i>
1	15	14	1,376 Bytes
2	20	19	1,836 Bytes
3	13	12	1,192 Bytes

Regular Volume Hierarchy				
<i>Image/Cut</i>	<i>Empty Vol</i>	<i>Full Vol</i>	<i>Data Pnts</i>	<i>Total Size</i>
1/25	10,820	1,347	13,836	104,012 Bytes
1/15	13,623(est)	2,000(est)	25,000(est)	162,492(est) Bytes
1/10	16,683(est)	3,000(est)	50,000(est)	278,732(est) Bytes
2/25	146	197	269	2,448 Bytes
2/15	234	278	350	3,448 Bytes
2/10	356	373	445	4,696 Bytes
3/25	1,794	1,581	17,388	83,052 Bytes
3/15	1,794	1,581	17,388	83,052 Bytes
3/10	2,456(est)	2,457(est)	300,000(est)	1.2M(est) Bytes

Adaptive Volume Hierarchy				
<i>Image/Cut</i>	<i>Nodes</i>	<i>Data Pnts</i>	<i>Hash Tbl</i>	<i>Total Size</i>
1/25	1,327	11,345	4,000	81,228 Bytes
1/15	2,600(est)	20,000(est)	4,000	146,400(est) Bytes
1/10	5,200(est)	35,000(est)	4,000	268,800(est) Bytes
2/25	37	141	4,000	5,452 Bytes
2/15	62	196	4,000	6,272 Bytes
2/10	114	258	4,000	7,768 Bytes
3/25	824	8,021	4,000	55,860 Bytes
3/15	1,776	11,681	4,000	93,348 Bytes
3/10	3,100(est)	20,000(est)	4,000	158,400(est) Bytes

Object/Volume Hierarchy					
<i>Image/Cut</i>	<i>Obj Nodes</i>	<i>Interior Vol</i>	<i>Leaf Vol</i>	<i>Data Pnts</i>	<i>Total Size</i>
1/25	15	53	278	3,067	20,420 Bytes
1/15	15	146	753	4,816	39,148 Bytes
1/10	15	296	1,569	6,948	67,596 Bytes
2/25	20	0	1	58	2,012 Bytes
2/15	20	0	1	58	2,012 Bytes
2/10	20	0	1	58	2,012 Bytes
3/25	13	188	1,110	8,994	63,832 Bytes
3/15	13	497	2,803	14,017	125,200 Bytes
3/10	13	953	5,656	21,677	223,844 Bytes

Table 4.5 Hierarchy Sizes

The size of the object hierarchy is the smallest of the four hierarchies. This result was expected due to the minimal structure used to build the object hierarchy.

The size of the regular volume hierarchy is strongly influenced by images which contain primitives of vastly different sizes. The scene in *Image 1* and the scene in *Image 2* both have artificial horizons. This means that there are some large objects, such as a sphere or a polygon, on which the majority of the objects are placed. The large objects are several orders of magnitude larger than the smaller objects in the scene. The initial volume of the regular volume hierarchy must enclose the entire scene, including these large objects. During subdivision many subvolumes are created before the small complex objects begin to be subdivided. Further, the large objects are subdivided into many small pieces by this process. The figures in Table 4.5 show how extensive this problem is. Nearly 90 percent of the subvolumes for *Image 1/25* (the 25 is the subdivision criteria) are empty. For the smaller subdivision rates (15 and 10) for *Image 1* there was insufficient stack space on the computer to complete the division process using the regular volume hierarchy. The values for the last two division levels were estimated by considering the size of the smallest object compared to the size of the initial volume and using data obtained from the subdivision of the other hierarchies. The regular volume hierarchy is the largest of the four hierarchies on *Image 1* due to the empty volumes and repeated subdivision of the large polygons. The size of the regular volume hierarchy was similar in size to the other hierarchies for *Image 2*. If *Image 2* was subdivided further (to a level of 5), the size of the regular volume hierarchy would start to grow quickly. There are no very large primitives in *Image 3* and the amount of space required to build a regular volume hierarchy for this image is similar to the other volume hierarchies. Not until the last subdivision level did the regular hierarchy begin to grow quickly. The last subdivision level for *Image 3* using the regular volume hierarchy had to be estimated due to stack overflows.

The adaptive volume hierarchy eliminates empty volumes by dividing nodes individually, based on the number of primitives inside them. The adaptive hierarchy is smaller than the regular volume hierarchy for the first image due to the elimination of the empty volumes and the reduction of the number of times the large polygons are subdivided. About half of the space saved is used to support the construction tree and hash table used to find volumes. Even with this reduction, the last two subdivision levels for *Image 1* and the last subdivision level of image *Image 3* had to be estimated due to stack overflows. The adaptive volume hierarchy does not show the pattern of sudden growth when the volume size becomes small that was apparent for the regular volume hierarchy.

The object/volume hierarchy is the most stable of the volume hierarchies. The stability and small size of the object/volume hierarchy is due to the subdivision procedure and the tendencies of objects to be compact and regular. Irregular objects will cause the performance of the object/volume hierarchy to degrade. The object/volume hierarchy is less than one third the size of the regular and adaptive volume hierarchies on *Image 1*. The object/volume hierarchy is also smaller than the regular and adaptive volume hierarchies on *Image 2*. The object/volume hierarchy is twenty five percent larger than the adaptive volume hierarchy on *Image 3*. This image has less open area than the other two test images and as such each hierarchy divided the data in a very similar manner. The object/volume hierarchy is larger in part due to the overhead required to store the bounding volumes at each node.

4.3.3. Rendering

The total time involved in producing a sequence of frames is dependent on the number of frames, the complexity of the scene, and the type of data hierarchy and traversal algorithm used. The total time for these operations is reflected in Equation 4.1.

$$T_T = \sum_{i=1}^{N_F} T_{F_i} + \sum_{i=1}^{N_H} T_{H_i} \quad 4.1$$

where,

T_T = Total system time.

N_F = Number of frames rendered.

T_{F_i} = The time required to render frame i .

N_H = Number of times the hierarchy is rebuilt.

T_{H_i} = The time required to rebuild hierarchy i .

The first term represents the rendering time whereas the second term is the time required to build the hierarchy. Historically the main research efforts have been directed to reducing the rendering time. This work correctly assumed that the construction of the hierarchy always takes less than ten percent the total rendering time. Continued refinements to the rendering algorithms and applications of special purpose hardware have decreased the rendering time to the point that the construction of the hierarchy cannot be ignored. The construction time of the four hierarchies is considered first.

4.3.3.1. Construction

The time required to construct a hierarchy is dependent on the scene being rendered. For volume based hierarchies even small changes in the scene, such as changing the size of a room, may cause objects to be subdivided which were not divided before. The lack of any formal method for classifying or evaluating the geometric complexity of a scene prohibits a complete analytical analysis of the time required to traverse each hierarchy. The three test images introduced in the previous section are used to substantiate the analysis presented next.

Each algorithm will construct the hierarchy from the initial linked list of simple objects as described earlier. The object hierarchy is considered first. Because the

construction of the object hierarchy is done manually, the time varies greatly. Equation 4.2 is the time required to build the object hierarchy.

$$T_O = \sum_{i=1}^{N_{C_i}} T_L + T_S + N_O T_M \quad 4.2$$

where,

T_O = Total object construction time.

N_{C_i} = Number of children of object node i .

T_L = The time required to build a link to a node.

T_S = The time required to arrange the object nodes.

N_O = Number of object nodes.

T_M = The time required to initialize the transformation matrices.

The time required to build the links between object nodes and the time required to initialize the transformation matrices is a fair estimate of the time required to update the object hierarchy between frames. The processing required to determine what updates to make between frames can be substantial, but this computation is independent of the hierarchy used and beyond the scope of this work.

The time required to create the object hierarchy for the three test images is listed in Table 4.6. Nearly all the time was spent in user think time. The time required to build the links between nodes and update the transformation matrices is very small, usually a few milliseconds.

The time required to build and update the object section of the object/volume hierarchy is similar to the object hierarchy just discussed. The following analysis of the volume based hierarchies will ignore the time required to build the object section of the object/volume hierarchy. This time will be included in the final analysis.

Volume Hierarchies				
<i>Image/Cut</i>	<i>Regular</i>	<i>Adaptive</i>	<i>Obj/Vol</i>	<i>Obj/Vol Update</i>
1/25	93.1s	86.7s	18.6s	2.4ms
1/15	160s(est)	150s(est)	32.8s	2.4ms
1/10	275s(est)	250s(est)	47.7s	2.4ms
2/25	.91s	.85s	.29s	3.2ms
2/15	1.46s	1.27s	.29s	3.2ms
2/10	2.71s	167.5s	.29s	3.2ms
3/25	71.2s	58.72s	58.5s	2.1ms
3/15	98.7s	93.8s	86.1s	2.1ms
3/10	160s(est)	140s(est)	120.4s	21.ms

Object Hierarchy				
<i>Function</i>	<i>Parameters</i>	<i>Image 1</i>	<i>Image 2</i>	<i>Image 3</i>
Object Const	$T_L = 3.6\mu s, T_M = 157\mu s$	<300s	<200s	<380s
Object Update	N_O, N_C see Tbl 4.5	2.7ms	3.2ms	2.6ms

Table 4.6 Construction Timings

The construction of the volume based hierarchies proceeds automatically and is dependent on the size, shape, and location of objects in the scene. The time required to construct a volume hierarchy is mainly dependent on the number of volume nodes created and the amount of primitive clipping that occurs. Equation 4.3 is the time required to build a volume hierarchy.

$$T_V = N_V T_L + \sum_{i=1}^{N_V} T_{C_i} + T_{AUX} \quad 4.3$$

where,

T_V = Total volume construction time.

N_V = Number of volume nodes.

T_L = The time required to build link nodes.

T_{C_i} = The time required to clip the data for node i .

T_{AUX} = The time required to build any auxiliary structure.

The first term of Equation 4.3 is the time required to build a link to each volume node created. This includes volume nodes which contain data and interior volume

nodes. The second term represents the clipping time. The time required to clip the data for each volume node is complicated by the use of a progressive clipping algorithm to subdivide the data and the fact that some of the volume nodes may be interior nodes. The last term includes the time required to build any auxiliary data structures such as hash tables and tree structures.

The time required to construct the regular volume hierarchy, the adaptive volume hierarchy, and the object/volume hierarchy are listed in Table 4.6. The timings for all three test images using the termination criteria of ten, fifteen, and twenty five primitives per volume is given for each hierarchy.

The time required to build the object hierarchy is the one of the longest of the four hierarchies. This is completely due to the time required to interactively construct the hierarchy. The time required to actually build links and reset transformations matrices is very small. The interactive construction occurs only once, at the start of an animation sequence. The time required to update the object hierarchy between frames is so small that the total construction time over many frames will be the smallest of all four hierarchies.

The time required to construct the volume hierarchies is very similar. Most of the time spent constructing the volume based hierarchies is used to clip the data. While one hierarchy may have more or larger volumes, the amount of actual data clipped is more or less the same. This factor accounts for the similar construction times, especially on *Image 3*. The construction time for the object/volume hierarchy for *Image 2* is less than the other hierarchies because no clipping at all was required. The design of the object/volume hierarchy resulted in a more compact structure being created for *Image 1* than was created by the other volume hierarchies. This resulted in lower construction times for the object/volume hierarchy.

Another key factor to observe is the object/volume hierarchy is not rebuilt

between frames of an animation sequence. This is due to the object based section at the top of the object/volume hierarchy. The update times for the object/volume hierarchy are listed in Table 4.6. As discussed earlier, the other volume hierarchies are rebuilt between most, if not all, the frames. The total time spent building a hierarchy for an animation sequence is examined after the analysis of traversal time is presented.

4.3.3.2. Traversal

The time required to traverse each hierarchy is dependent on the scene being rendered and the ability of the hierarchy to quickly eliminate large portions of the data from consideration. The absence of a method of classifying and evaluating the geometric complexity of images prohibits a complete analytical analysis of the relative traversal times. The test images introduced earlier are used to substantiate the following analysis.

The traversal of the object hierarchy is considered first. The time required to traverse the object hierarchy is dependent on the number of object nodes visited and the number of data primitives tested. The time required to test an object node is the sum of the time required to test the bounding volume and to follow the link to the child. The time to test a primitive is dependent on the type of primitive being tested. Equation 4.4 is the time to traverse the object hierarchy.

$$T_O = N_O (T_B + T_L) + N_P T_P \quad 4.4$$

where,

T_O = The time to traverse the object hierarchy.

N_O = The number of object nodes visited.

T_B = The time to test a bounding volume.

T_L = The time to traverse a link.

N_P = The number of primitives tested.

T_P = The time to test a primitive.

Table 4.7 contains the time required to traverse the object hierarchy for the three test images. The time to test the bounding volumes is quite small. Most of the traversal time is spent intersecting data primitives. The values in Table 4.10 are per ray averages.

Object hierarchies perform best when the simplest objects contain very few primitives and only one node at each level passes the bounding volume test. This is the case for most of the rays traced in *Image 2*. The value of N_O is larger but few of the visited object nodes were ever expanded. The average time to trace a single ray is the best of all the hierarchies on this test image. In *Image 3* the objects are large and overlapping causing many objects to pass the bounding volume test at each level. Due to the large number of primitives tested per ray the timing values for this image are very poor.

Each of the three volume hierarchies have strengths which reduces some part of the traversal time. The regular volume hierarchy reduces the overhead to find any volume to a simple and direct calculation. If an intersecting primitive is found in the first volume tested, this method can be considered a constant time algorithm. The adaptive volume hierarchy assumes that a few rays may travel through several empty volumes before entering a volume with data and intersecting a primitive. The adaptive system eliminates most of these empty volumes at the added cost of traversing a simple tree to find the next volume. The object/volume hierarchy assumes that a few rays will travel through some empty volumes and some volumes with data before entering a volume where it strikes a data primitive. The object/volume hierarchy eliminates most of the empty volumes and prunes volumes that contain data the ray does not intersect using bounding volumes. Based on these observations a scene can be constructed such that any one of these hierarchies will out perform the others. The three test images were constructed to be representative of actual images and not to show the

strengths of a particular algorithm.

Object Hierarchy					
Image	N_O	$(T_R + T_L)$	N_P	T_P	Total Time
1	17.97	.153ms	260	.303ms	81.5ms
2	20.0	.153ms	4.92	.303ms	4.55ms
3	16.9	.153ms	712	.303ms	218ms

Regular Volume Hierarchy						
Image/Cut	N_V	T_N	N_P	T_P	T_O	Total Time
1/25	10.4	.102ms	314	.303ms	0	96.2ms
1/15	11.0(est)	.102ms	265(est)	.303ms	0	81.4(est)
1/10	13.0(est)	.102ms	200(est)	.303ms	0	61.9(est)
2/25	4.18	.102ms	42.1	.303ms	0	13.2ms
2/15	5.33	.102ms	39.6	.303ms	0	12.5ms
2/10	6.17	.102ms	36.7	.303ms	0	11.7ms
3/25	4.39	.102ms	32.3	.303ms	0	10.2ms
3/15	5.17	.102ms	29.6	.303ms	0	9.49ms
3/10	6.75(est)	.102ms	27.0(est)	.303ms	0	8.86ms(est)

Adaptive Volume Hierarchy						
Image/Cut	N_V	T_N	N_P	T_P	T_O	Total Time
1/25	42.1	.216ms	296	.303ms	.05	98.8ms
1/15	60(est)	.216ms	250(est)	.303ms	.05(est)	88.8(est)
1/10	80(est)	.216ms	180(est)	.303ms	.05(est)	71.9(est)
2/25	2.84	.216ms	39.4	.303ms	.05	12.6ms
2/15	4.27	.216ms	37.4	.303ms	.05	12.3ms
2/10	6.91	.216ms	31.0	.303ms	.05	10.9ms
3/25	40.7	.216ms	20.1	.303ms	.05	14.9ms
3/15	49.8	.216ms	18.8	.303ms	.05	16.5ms
3/10	60.0(est)	.216ms	17.0(est)	.303ms	.05(est)	18.2(est)

Object/Volume Hierarchy						
Image/Cut	N_V	T_N	N_P	T_P	T_O	Total Time
1/25	24.7	.245ms	54.0	.303ms	2.75ms	25.4ms
1/15	36.9	.245ms	43.2	.303ms	2.75ms	24.8ms
1/10	55.1	.245ms	33.6	.303ms	2.75ms	26.4ms
2/25	1	.245ms	4.92	.303ms	3.06ms	4.80ms
2/15	1	.245ms	4.92	.303ms	3.06ms	4.80ms
2/10	1	.245ms	4.92	.303ms	3.06ms	4.80ms
3/25	43.2	.245ms	19.8	.303ms	2.59ms	19.2ms
3/15	54.5	.245ms	17.1	.303ms	2.59ms	21.1ms
3/10	67.1	.245ms	15.0	.303ms	2.59ms	24.2ms

Table 4.7 Traversal Timings

Figure 4.6 show three rays entering a subdivided object. Ray 1 strikes an object in the first volume tested, ray 2 travels through a few empty volumes before striking an object or leaving the hierarchy, and ray 3 pass through empty and full volumes before striking an object or leaving the hierarchy. The relative performance of the volume hierarchies is dependent on the number of occurrences of these three types of rays. Due to the time required to intersect a ray with a primitive, there need not be many occurrences of ray type 3 to justify the use of the object/volume hierarchy.

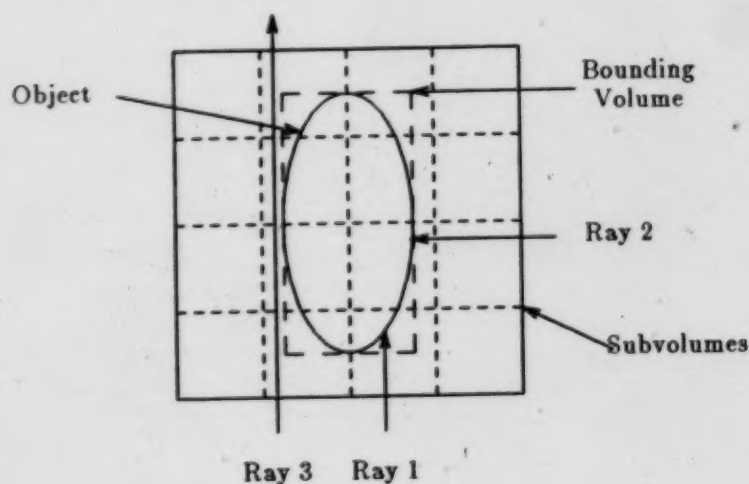


Figure 4.6 Volume Culling

The time required to traverse the volume based hierarchies is dependent on the number of volumes visited, the number of primitives tested, and the overhead incurred while traversing the hierarchy. Equation 4.5 is the time required to traverse the volume hierarchies.

$$T_V = N_V T_N + N_P T_P + T_O \quad 4.5$$

where,

T_V = The time to traverse the volume hierarchy.

N_V = The number of volume nodes visited.

T_N = The time required to determine the next volume.

N_P = The number of primitives tested.

T_P = The time to test a primitive.

T_O = The time to perform any overhead.

The timings for the three test images with ten, fifteen, and twenty five primitives per volume as the cutoff point during construction is listed in Table 4.7. The values are averages on a per ray basis.

The optimal performance of the regular volume hierarchy was never quite realized due to how large the hierarchy grew as the volume size was decreased. The time to find a volume for the regular volume hierarchy is reflected in the term T_N . This hierarchy requires no further operations to find a volume and as such the time to perform the overhead is zero. The performance of this hierarchy on *Image 3* was the best of the four hierarchies. This was due to the large number of closely spaced primitives in the image and the fact there was little empty space. The performance of the regular volume hierarchy on *Image 1* was quite poor. This was because a large number of primitives were compared to each ray. Many of the rays passed through volumes but did not intersect an object (type 3, Figure 4.6). Even with further subdivision it is doubtful that the performance of the regular volume hierarchy would match that obtained by the object/volume hierarchy.

The adaptive volume hierarchy never out performed all of the other hierarchies on a particular image. The timing values for the adaptive volume hierarchy compared favorably to the best timing value for the other hierarchies on all images but *Image 1*. The cause of this poor performance was due to rays passing through volumes which contained data but where the ray did not intersect a primitive.

The key factor to notice for the object/volume hierarchy is that in all cases the

number of primitives tested per ray is always less than or equal to the other hierarchies. If complex data primitives are used, then an object/volume hierarchy is guaranteed to out perform all the other hierarchies on nearly any image. With the simple primitives used in the three test images, the object/volume hierarchy out performed the other hierarchies on *Image 1*. This was directly due to the culling, by the bounding volumes, of volumes through which the ray passed but did not intersect the object. The performance of the object volume hierarchy on the other images compares very favorably with the best times obtained for that image. This is especially true for *Image 2* where the performance was nearly identical to the object hierarchy.

The total system time over hundreds of frames clearly favors the object/volume hierarchy. The hierarchy need not be rebuilt between frames saving the construction time incurred by the regular volume hierarchy and the adaptive volume hierarchy. It is also efficient to traverse on most images, unlike the object hierarchy. For a low resolution image of 128 pixels by 128 pixels the time required to render the image is about the same as the time required to construct the hierarchy. This makes the construction of the hierarchy between frames a major concern. For high resolution images, the construction time is negligible unless the special purpose architecture, presented in the next section, is used. This architecture is considered next.

4.4. Ray Tracing Architectures

The models and algorithms presented in this and previous chapters improve the overall performance of the ray tracing system, but further improvements in rendering efficiency are necessary. These improvements are realized by the application of special purpose architectures.

4.4.1. Introduction

A multiprocessor ray tracing system that is based on a new parallel ray tracing algorithm is introduced. This ray based system offers greater rendering efficiency over existing parallel ray tracing systems by improving processor usage and minimizing interprocessor communications. Before presenting this structure the resource requirements of the ray tracing system is examined. This analysis includes all the rendering processes such as shading, mapping, and I/O.

4.4.2. Resource Requirements

The computer resources required by ray tracing systems are examined in three groups. These are processor requirements, storage requirements (both disc and main memory), and I/O requirements.

The resource requirements were determined experimentally using the algorithms and implementation techniques described in the previous chapters. These algorithms have been shown to produce superior images more efficiently than other ray tracing systems.

4.4.2.1. Processor Requirements

Table 4.8 contains a profile of the amount of processor time spent on all operations of the rendering process that require more than one half of one percent of the total processor time. The three test images introduced in the previous sections were used to obtain these values. Each image was rendered at a resolution of 256 by 256 pixels with no anti-aliasing. The volume size was set to a maximum of 15 primitives per volume to obtain maximum rendering efficiency using the object/volume hierarchy. All code was written in the C programming language and executed on a VAX 11/780 with a floating point accelerator. The timing values were obtained using a program called *gprof* written by S. Graham, P. Kessler, and K. McKusick [21]. The timing

values presented in Table 4.8 include all the rendering operations and system overhead. These value will be slightly larger then the timing values presented earlier due to the inclusion of the system overhead.

Routine	Image 1		Image 2		Image 3		Avg(sec)
	Calls	Time(s)	Calls	Time(s)	Calls	Time(s)	
<i>ptrace</i>	783868	1256	254802	131	578364	1293	.00182
<i>qtrace</i>	0	0	272242	252	0	0	.00092
<i>otrace</i>	2410506	430	4166057	632	2876956	467	.00019
<i>vtrace</i>	1542403	296	534051	122	2139110	467	.00017
<i>ndmult</i>	1719298	244	1068102	152	1225832	175	.00014
<i>sqrt</i>	277821	66	738251	175	344267	82	.00024
<i>render</i>	1	41	1	41	1	41	41
<i>wpizel</i>	66049	41	66049	41	66049	41	.00062
<i>teval</i>	66247	40	105615	65	90413	47	.00058
<i>shade</i>	66247	35	105615	67	90413	55	.00060
<i>mknode</i>	95655	19	114553	26	1465596	30	.00021
<i>readd</i>	1	2	19	.1	1	5	.28437
<i>mmult</i>	47	.12	83	.18	132	.31	.00217
<i>fpizel</i>	1	1	1	1	1	1	1
<i>misc</i>	na	260	na	267	na	480	na
Total	45m 30s(2730)		32m 53s(1973)		52m 59s(3179)		43m 47s

Table 4.8 Processor Timings

The function of each of the routines listed in column one are described next and summarized in Table 4.9.

The intersection of rays with primitives is divided into two main routines. The routine *ptrace* intersects a ray with all the polygons in a leaf volume node. The routine *qtrace* intersects a ray with all the quadric surfaces and their bounding planes in a leaf volume node. The function *sqrt* is called by *qtrace* to perform a square root operation as part of the intersection calculation of the ray with each quadric surface.

The time required to traverse the object/volume hierarchy is reflected in several routines. The time to traverse the object section of the hierarchy is represented by the routine *otrace*. The time required to traverse the volume section of the hierarchy is

reflected in the routine *vtrace*. These times include the time required to intersect the bounding volumes. The time required to normalize the ray to the local coordinate system at each node is reflected in the routine *ndmult*.

The routine *render* is responsible for generating incident rays while the routine *shade* is responsible for generating all secondary rays. The routine *shade* calls *mknode* as part of the shadow testing process. The routine *teval* evaluates the virtual lighting tree created by the ray tracing process to obtain the partial pixel values. The routines *render*, *shade*, and *teval* call the *sqr* routine to perform part of their function. The routine *wpixel* collects partial pixel data into pixel values. The time required to perform dozens of other operations required by the ray tracing system is included under *misc*. This includes routines for smooth shading, bump mapping, texture mapping, and system initialization.

Table 4.9 summarizes the list of routines shown in Table 4.8 and specifies their function.

Routine	Description
<i>ptrace</i>	Intersects a ray and all the polygons in a leaf node.
<i>qtrace</i>	Intersects a ray and all the quadric surfaces in a leaf node.
<i>otrace</i>	Traverses the object hierarchy.
<i>vtrace</i>	Traverses the volume hierarchy (see also <i>ndmult</i>).
<i>ndmult</i>	Normalizes a ray to local coordinates.
<i>sqr</i>	A square root function.
<i>render</i>	Generates the initial rays.
<i>wpixel</i>	Averages subpixel values and output final pixel values.
<i>teval</i>	Traverses the lighting tree.
<i>shade</i>	Generates all secondary rays and builds light tree.
<i>mknode</i>	Obtains memory and initializes nodes.
<i>readd</i>	Read image data.
<i>mmult</i>	Performs matrix multiplication.
<i>fpixel</i>	Dumps final raster to the display or file.

Table 4.9 Rendering Routines

Table 4.10 presents a summary of the operations listed in Table 4.8. These

figures show that the intersection calculations and traversal of the hierarchy persist as the most computationally intensive task. Included in Table 4.10 are the effective number of primitives in each image before and after the construction of the hierarchy.

Function	Image 1		Image 2		Image 3	
	Time(s)	%	Time(s)	%	Time(s)	%
<i>intersection</i>	1256	46%	448	23%	1293	41%
<i>traversal</i>	970	36%	906	50%	1091	34%
<i>others</i>	504	18%	619	32%	794	23%
Primitives						
<i>initial</i>	1,638		58		2,708	
<i>final</i>	4,816		58		14,017	

Table 4.10 Timing Summary

4.4.3. Storage Requirements

Table 4.6 lists the amount of space required to store the object/volume hierarchy for each of the three test images. This includes all of the storage for headers and links. The storage requirements for the rendering code is also given.

Node/Image	Total Size(bytes)
Image 1	247,584
Image 2	10,896
Image 3	411,328
code	194,416
data overhead	15,796
raster	196,608

Table 4.11 Hierarchy Storage Requirements

The ray tracing code requires little space and has modest storage requirements. A copy of the raster is normally kept in memory at all times to allow the ray tracing system quick access to the raster for the filtering operations of anti-aliasing. The space required for the raster is dependent on the resolution of the image. The size of the raster listed in Table 4.11 is 256 by 256 by 3 bytes.

The amount of space required to store the object/volume hierarchy and bump and texture maps can become a problem if there is insufficient main memory available in the computer system. Fortunately, not all the data need be present in main memory during the entire rendering process to obtain satisfactory results. The sections of the hierarchy traversed by a given ray are similar to the sections visited by subsequent rays. The sections of the hierarchy common to two subsequent rays and the total percentage of the hierarchy traversed by a given ray is dependent on the image coherence in the scene being rendered and the pattern in which the rays are traced. Table 4.12 shows the average amount of active memory required during the rendering process for the three test images. These values were obtained over ten second intervals. On the average, 10K bytes of new pages were required every 60 seconds. The rays are traced systematically starting at the lower left of the display and proceeding up the display on a single z subpixel line. This procedure is repeated for each z subpixel line from the left end of the screen to the right.

Image	Total Size	Active Size
1	654,404	73,281
2	417,716	66,332
3	818,148	101,918

Table 4.12 Active Pages

This coherence also holds for texture and bump maps. These maps can be large, as described in Chapter 2, but because the maps affect only certain areas of the screen at a given time, only the sections of the map required for the general area being ray traced need be present in main memory.

4.4.3.1. I/O Requirements

The I/O requirements of the ray tracing system involve the loading of the data hierarchy and appropriate maps into memory and the final output of the raster to a frame buffer or file. The size of the input data for the three test images is listed in Table 4.11. The amount of time required to read in the data and output the raster image, is listed in Table 4.8. The time to read the data is reflected in the routine *readd* and in the routine *mknodc*. The reading of the hierarchy involves the allocation of memory and building links between nodes. An improved implementation of the routines for inputting the data, which include a memory management system, would improve the efficiency of this process. The time required to output the raster is reflected in the routine *fpizel*. These figures show that less than one percent of the rendering time is spent performing I/O. If desired, the output of the raster can be done one pixel at a time. This would require three bytes of I/O about every 40ms.

4.4.4. Multiprocessors

Improving the efficiency of the hardware components can only improve the speed of the rendering process up to a point. Further, and possible extensive speed up, will occur with the use of multiprocessor systems. The key to how many processors can be applied to the problem is dependent on how the ray tracing task is divided. This section will present a parallel ray tracing algorithm that divides the rendering task by rays. It will be shown that dividing the rendering task by rays has enough granularity to obtain nearly optimal processor utilization. It will also be shown that effective implementations exist for nearly any number and size of processing nodes.

4.4.4.1. Parallel Algorithm

The two classical bottlenecks encountered when multiprocessor systems are used to improve the performance of a given application are excessive interprocessor communications and lack of enough problem granularity to apply the required number of processors to the task. The ray tracing algorithm presented earlier possesses the necessary properties to overcome these bottlenecks.

Incident rays are the initial rays that originate at the location of the viewer and pass through the corner of each pixel on the display. The results obtained by tracing a given incident ray, and all secondary rays that are generated, are completely independent of the results obtained from other incident rays. The number of incident rays required to produce a satisfactory image ranges from just over a quarter million to several million. The ray tracing algorithm introduced in the previous section is converted to a parallel ray tracing algorithm by distributing the tracing of incident rays among several processors.

By dividing the rendering task by incident rays, as few as one ray could be assigned to each processor. This would utilize millions of processing elements in parallel for a single image. While tracing an incident ray, no communication is required between any processors. The only communication required is the assignment of rays and the final collection of results.

Conceptually, an incident ray and the secondary rays that are generated can intersect any primitive in the scene. This requires that each processing node have access to all the image data. If the processing nodes are large enough, all the image data can be stored at each node. If the node is not of sufficient size, the locality of memory reference property of the ray tracing algorithm can be exploited by one of several methods to share the data between the processing nodes.

4.4.4.2. Multiprocessor Architecture

A multiprocessor ray tracing system using the parallel ray tracing algorithm described earlier is proposed. It is assumed that each processing node has enough primary and secondary memory to store the entire description of the scene, rendering code, and associated system support routines. The use of small processing nodes is not addressed by this thesis. A host computer is used to control the multiprocessor system. The host computer is the source of the image data and rendering code, and the destination of the final pixel values. The purpose of this section is to show the power of the parallel ray tracing algorithm and does not attempt to completely define a multiprocessor system.

The basic layout of this system is shown in Figure 4.7. A global communications system, such as a simple net, is used for all communications. It is shown that this simple communications system can support the communication needs of a large number of processing nodes. The precise number of processors that can be supported before exceeding the bandwidth of the net is determined shortly.

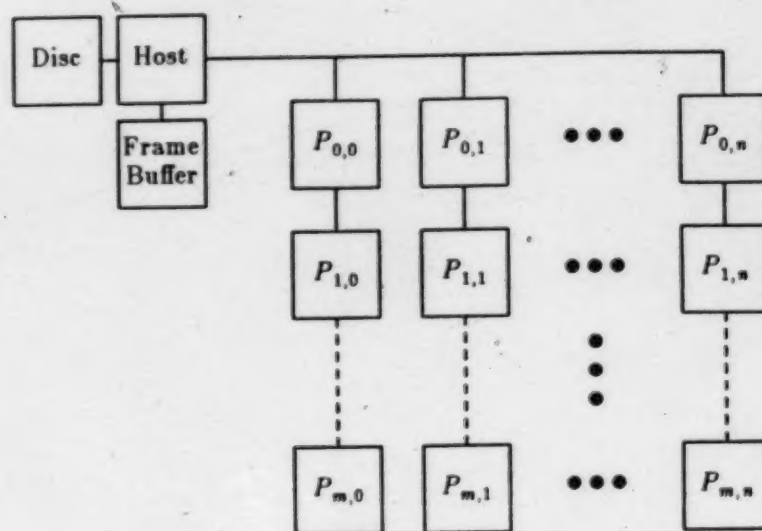


Figure 4.7 Multiprocessor Architecture

Four simple commands are used to control the entire system. Each command is 16 bits long. The first two bits are the command code and the last 14 bits are the processor ID. With this format 16,384 processors can be individually addressed. The four commands are *load code*, *load data*, *load display polygon*, and *return result*.

Each processor has a small amount of code located in read only memory to recognize the *load code* command. The host initializes each processor node by sending the *load code* command followed by the rendering code. The format of the rendering code begins with a four byte word count to specify the length of the code, then a four byte address to specify where the code should be loaded, and finally the code. The rendering code is common for each processor and as such the host broadcasts the code in parallel to all the nodes. Once the code is loaded, each node begins execution starting at the address where the code was loaded. Using an 8 Megabit net and the rendering system described earlier, the code initialization phase will transmit 194,424 bytes of rendering code in .194 seconds. There is no contention for the net during this phase of the operation.

Next the data is loaded into each processor node by the host. The host transmits the *load data* command and then the image data. The format of the data is the same as the code format. A four byte count is sent first to specify the length of the data followed by four byte address to specify where the data is to be placed. This data is also common to all processors and can be loaded in parallel. Included in this data is the display polygon and the resolution of the system. For discussion purposes it is assumed that Image 1, which was described earlier, will be rendered. The loading of the 247K bytes of image data will require .247 seconds. There is no contention for the net during this phase of the operation.

Next, each processor is assigned a section of the initial display polygon to render. The display polygon is described parametrically as discussed in Section 2.3. The data required to specify a section of the display polygon requires parametric values of the upper left and lower right hand corners of the subdisplay polygon. Two bytes are used for each value, thus setting the maximum image resolution to 65,536 by 65,536 pixels. The host processor sends a *load display polygon* command to each processing node one at a time. The ID section of the command is used to indicate the target processor. The total message length is 10 bytes. Assume that there are N processing nodes, then the time required to assign work to all the processors is .00001 N seconds. Actually, if the number of processors is known, and each processor is assigned a particular position (first, second, third etc.) then each processor can determine the size and location of its own subpolygon. This command will be included for clarity due to the small amount of overhead it requires, but will be dropped later.

Assume for the moment that the image is to be rendered at 1024 by 1024 pixels and that each section of the image is as complex as another. These assumptions are examined in detail later. This means all processors will finish at about the same time, offset only by the different times at which they were assigned their section of the display polygon to render. When a processor finishes a tracing a ray, the net is used to

send the final pixel values to the host. The processor node checks to see if the net is free and issues a *result* command. The format of the pixel is two bytes for each pixel coordinate and three bytes for the pixel value. The total message length is 9 bytes which requires .000009 seconds to transmit.

The transmission of results back to the host will cause contention on the net. Clearly, groups of results should be sent to reduce the chance of collisions. A faster net and a packet protocol can also be employed. These questions are beyond the scope of this thesis. It is assumed that there are no collisions or errors and the results are sent back one at a time. It is not difficult to design an actual system using a faster net and larger packets which will exceed the total bandwidth of this idealized system.

Each processor can trace a ray for this image in .04 seconds as shown in Table 4.8. Under optimal conditions 4,444 processors can transmit pixel values over the net in .04 seconds. The total time to render an image under these conditions without exceeding the bandwidth of the communications system is given by Equation 4.6.

$$T = T_C + T_D + N T_P + \frac{D_R}{N} T_R \quad 4.6$$

where,

T = Total rendering time.

T_C = Time to transmit the code.

T_D = Time to transmit the data.

T_P = Time to transmit the pixel data.

D_R = The resolution of the image.

T_R = Time to trace a ray.

N = The number of processors.

Using the values obtained earlier, the total time to produce the first image of a animation sequence with 4,444 processors is 9.92 seconds. This is a speed up of 4228

times. The subsequent images of the animation sequence do not require the data or code to be retransmitted and the updates between frames were sent with the initial data. The time required to produce these images is 9.4 seconds which is a speed-up of 4444 times.

The host system is required to simply route the incoming pixel values into the frame buffer. The rate at which the pixels are received with 4444 processor nodes is one every 9 microseconds. This is well within the capabilities of current computer technology. If a faster net is used, and more processors added, the loading of the frame buffer will become a bottleneck in the system. This should occur before the number of processors is increased a full order of magnitude.

If the scene is larger, the time to transmit the image data will increase. An image containing a 100 million bytes would require 100 seconds to transmit. As the complexity of the image increases the time to trace a single ray slowly increases. Image 3 contains over 100 times the number of primitives as Image 2 yet the time to trace each ray increased only 1.3 times. An interesting property of this architecture is that as the time required to trace a ray increases, each processor will put less demand on the net, and hence more processors can be applied to the task without overloading the net.

There may be some unevenness in the manner in which a processor node returns pixel values which can cause contention on the net. This effect is due to the complexity of the image being slightly different from one pixel to the next. A several pixel buffer at each processor node may be required to keep the throughput constant. With a maximum processor system, a slightly larger buffer may be necessary.

It was assumed that each processor would complete its task at about the same time. Images do exist that are simple in one section but complex in others. The performance of the parallel ray tracing algorithm was shown earlier to be nearly independent of image complexity and as a result, changes in the geometric complexity of the

image will not drastically change the time required to trace a ray. If the lighting complexity of the image is different from one section of the scene to another there can be a major difference in the time required to trace an incident ray to completion in different parts of the scene.

In the worst circumstances, one processor can take much more time to complete its task than the other processors. There are two methods to reduce the chances of this occurring. One takes advantage of image coherence and the other takes advantage of frame to frame coherence. The methods can be used together, but for clarity will be described individually. The frame to frame coherence method is presented first.

When a processor finishes a subpolygon it can immediately start on the second frame of the animation sequence. Once the host has received all the data it expects from a processor for a given frame, it can assume the processor will automatically start on the next frame. Rather than tracing the same subpolygon, the processor will do the subvolume directly to the right. This can be directly calculated from the previous subpolygon by assuming that each processor received the same size piece. If the current subpolygon was on the right edge of the main display polygon the processor indexes down a row and over to the left. If the current subpolygon is in the bottom right corner of the main display polygon the upper left hand corner should be used. During the rendering of an entire animation sequence, if there is a particularly difficult section of the image, each processor will have rendered part of this section an equal number of times, thus more evenly distributing the processing load. This technique is only effective if the image is similar from one frame to the next. It should be noted that if the image is not too similar from one frame to the next, then the changes in the images itself will help even the load. It would be most unfortunate if the complex part of an image happened to follow a particular processor as it moved from frame to frame thus reducing the total efficiency of the system, but this is not seen as a difficulty. This

scheme incurs no communication overhead and negligible processor overhead.

The second method divides the initial display polygon into $64/N$ subpolygons, where N is the number of processors. Each processor is assigned a set of 64 subpolygons to render. The value of 64 was chosen by assuming that the slowest processor is rarely more than 64 times slower than the fastest processor. When a processor has completed one subdisplay polygon from this set the next subdisplay polygon is automatically determined and started. The distribution of the set of subpolygons rendered by a processor is such that they are evenly distributed across the scene. This insures that if one particular part of the image is complex, all processors will render some of part the complex area equalizing system load. This technique takes advantage of image coherence, that is, the observation that an image changes little from one point to the next. Again, it would be most unfortunate if the complex parts of an image happened to be in all the locations a processor rendered but this is not seen as a difficulty when combined with the above frame to frame coherence technique. This scheme also incurs no communication overhead and has negligible processor overhead.

4.5. Summary

An object/volume data hierarchy, was introduced in this chapter which was shown to be smaller and easier to render for scenes with large open areas. Based on this hierarchy a parallel ray tracing algorithm was presented which divided the rendering task by rays. It was further shown with a simple multiprocessor architecture that the new parallel ray tracing algorithm has sufficient granularity and minimal communication overhead to effectively apply a large number processing elements.

Chapter 5

Conclusions

5.1. Introduction

The initial intent of this research was to study architectures for graphics systems. It was determined early that the scope of this research would be limited to ray tracing machines. This approach was taken in part due to the power of the ray tracing algorithm and in part because of the substantial amount of existing research on scanline rendering systems.

An important step in this research was to examine in detail, existing graphics techniques and ray tracing algorithms. To obtain a deep understanding of these algorithms, and to build a testbed in which simulations could be designed, a ray tracing based rendering package was implemented. Such a task is nontrivial, thus each algorithm was carefully analyzed before implementation. Several considerations were made before an algorithm was chosen based on the original goal of this research.

- Can the algorithm be divided between multiprocessors with minimal communications?
- Is the algorithm able to be divided into small enough pieces such that an arbitrary number of processors can be assigned to the task?
- Will the algorithm execute in parallel with other processes of the rendering system?
- Can the algorithm be divided into many small repetitive tasks for possible implementation in VLSI?

It was found that the algorithms described in much of the published work did not meet these criteria. As a result, several important areas were examined and new models and algorithms obtained. The next three sections summarize these

contributions.

Also during this process many minor algorithms and techniques were developed when existing solutions were unsatisfactory or where there was no published work on the subject. The most important of these is a smooth shading model for polygon meshes and an intersection algorithm for polygons.

The smooth shading algorithm introduced in Chapter 2 uses an additional normal located at the approximate center of the polygon to determine the normal at an arbitrary point on the surface of the polygon. The value of this normal is the average of the normals at all the nodes of the polygon. The method presented for obtaining the normal of an arbitrary point on the polygon uses this center normal in all calculations and as such distributes the curvature of the polygon in a more even manner than previous algorithms. This method also uses the normal found along the edge that is in line with the center normal and the intersection point. Hence, the normal values are completely independent of the location of the polygon in space. This solves a standing problem of varying shading during animation sequences. The calculations for the new algorithm are both straightforward and simple.

The intersection algorithm for polygons introduced in Section 2.3 is valid for polygons with an arbitrary number of sides. The calculation is much simpler than the method used for polygons in parametric form. By projecting the three dimensional problem to a two dimensional surface, substantial computation is saved. The calculations performed for each edge of the polygon are identical and simple. This suggests a VLSI implementation would be feasible.

5.2. Lighting Model

The lighting model presented in Chapter 3 models the diffuse, reflective, and transparent lighting components of a surface in a simple and homogeneous manner. The model also accurately includes lighting contributions from all light sources in the intensity calculation. This was demonstrated by using the model to include contributions from secondary light sources in the diffuse lighting calculations. Within the constructs of this model other lighting effects such as internal illumination can also be included. Previous lighting models do not accurately include lighting contribution from all light sources and they model each lighting component of the surface in a different manner. This model is also simpler and more general than the previous models used to include global lighting contributions in the diffuse lighting calculation.

The model was implemented using a distributed ray tracing algorithm, employing several techniques to reduce the number of rays traced. By using a ray tracing algorithm, the model can be applied to any scene and primitive description amendable to standard ray tracing algorithms. Because ray tracing is a point sampling technique, image errors can become apparent if the image is under sampled. The model minimizes over sampling by independently varying the sampling rate for each lighting component of the surface.

5.3. Object/Volume Hierarchy

The data hierarchy presented in Chapter 4 is a more efficient structure to traverse for a scene with large open spaces than existing proposals. It is a two-level structure. The top level is an object based hierarchy that gives the entire structure robustness by the use of transformation matrices at each node. The object section of the hierarchy is also a graph where common objects are shared thus reducing the size of the hierarchy.

The bottom sections of the hierarchy are volume based. An actual hierarchy is constructed and traversed in such a manner that the leaf nodes, containing the image

data, are considered in the order the ray would encounter them in a standard volume hierarchy. By considering the nodes in this manner the traversal can be terminated when the first intersecting primitive is found. The volume hierarchy is traversed such that bounding volumes can be used to prune large portions of the tree from consideration. This is a more effective procedure than conventional volume hierarchies when the ray passes through many volumes but does not strike an object.

5.4. Ray Based Architectures

A ray based parallel ray tracing algorithm was presented in Chapter 4. This algorithm possesses several properties that make it a prime candidate for a multiprocessor implementation. It was shown with a simple example that the communications required between large processing nodes executing this algorithm are almost negligible. As such, thousands of processors can be applied to the task using only a simple communications system. It was also shown that the processing load can be evenly distributed among processors with little overhead by taking advantage of image and frame coherence.

5.5. Further Work

Several interesting problems, which could be the topic of further studies, became apparent during the course of this research. The lighting model presented in Chapter 3 can be expanded to include prismatic lighting effects. This would present some interesting implementation difficulties due to the vast number of rays that would be required to carry the light frequency information. Several interesting possibilities to attack this problem include multiple source ray tracing and beam tracing. Also, the generality of the lighting model can be tested by including other lighting effects and determining their impact on the implementation techniques.

The design of a ray tracing architecture, based on the algorithms described in this

thesis, contains several areas which can be researched further. The parallel ray tracing algorithm was shown to require a very small working data set at any give time. This property can be exploited to apply small processing nodes to the task and share data between nodes. Where the image data is stored and how it is shared between processors can be a topic of further work. Possible solutions include processor arrays, shared disks, and multiple nets.

As the number of processors which can be applied to the task increases, the bottleneck at the frame buffer becomes an important factor. Further work is required to distribute the frame buffer between processors or on multiple nets.

References

- [1] J. Amanatides, "Ray Tracing with Cones", *ACM SIGGRAPH*, Vol. 18, No. 3, July 1984, pp. 129-136.
- [2] A. Appel, The Notion of Quantitative Invisibility and The Machine Rendering of Solids, *Proceedings ACM National Conference*, 1967, pp. 387-397.
- [3] J. F. Blinn and M. E. Newell, "Texture and Reflection in Computer Generated Images", *Comm. ACM*, Vol. 19, No. 10, Oct 1976, pp. 524-547.
- [4] J. F. Blinn and M. E. Newell, "Clipping Using Homogeneous Coordinates", *Proceedings of SIGGRAPH 77 in Computer Graphics*, Vol. 12, No. 3, July 77, pp. 245-251.
- [5] J. Blinn, "Models of Light Reflection for Computer Synthesized Pictures", *Proceedings. SIGGRAPH 77 in Computer Graphics*, Vol. 11, No. 2, August 1977, pp. 192-198.
- [6] J. Blinn, "Simulation of Wrinkled Surfaces", *Proceedings. SIGGRAPH 78 in Computer Graphics*, Vol. 12, No. 3, August 1978, pp. 268-292.
- [7] Phong Bui-Tuong, "Illumination for Computer Generated Images", *Comm. ACM*, Vol. 18, No. 6, June 1975, pp. 311-317.
- [8] L. Carpenter, L. Fournier and D. Fussell, "Fractal Surfaces", *Comm. ACM*, Vol. 25, No. 6, June 1982, pp. 371-384.
- [9] J. H. Clark, "Hierarchical Geometric Models for Visible Surface Algorithms", *Comm. ACM*, Vol. 19, No. 10, Oct 1976, pp. 547-554.
- [10] J. Cleary and B. Wyvill, *A Raytracing Machine*, Tech Report - University of Calgary, Calgary, Alberta, 1984.

- [11] M. Cohen and D. Greenburg, "The Hemi-Cube: a Radiosity Solution for Complex Environments", *ACM SIGGRAPH*, Vol. 19, No. 3, July 1985, pp. 31 - 40.
- [12] R. L. Cook, T. Porter and L. Carpenter, "Distributed Ray Tracing", *ACM SIGGRAPH*, Vol. 18, No. 3, July 1984, pp. 137-145.
- [13] F. Crow, "The Aliasing Problem in Computer-generated Shaded Images", *Comm. ACM*, Vol. 20, No. 11, November 1977, pp. 799-805.
- [14] F. Crow, "Shadow Algorithms for Computer Graphics", *Proceedings. of SIGGRAPH 77 in Computer Graphics*, Vol. 11, No. 2, July 77, pp. 242-247.
- [15] M. Dippe and E. Wold, "Antialiasing Through Stochastic Sampling", *ACM SIGGRAPH*, Vol. 19, No. 3, July 1985, pp. 69 - 78.
- [16] T Duff, "Smoothly Shaded Renderings of Polyhedral Objects on Raster Displays", *Proceedings. SIGGRAPH 79 in Computer Graphics*, Vol. 13, No. 2, August 1979, pp. 270-275.
- [17] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, Reading, Massachusetta, 1982.
- [18] A. Glassner, "Space Subdivision for Fast Ray Tracing", *Computer Graphics and Applications*, Vol. 11, No. 9, October 1984, pp. 15 - 22.
- [19] C. Goral, K. E. Torrance, D. P. Greenberg and B. Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces", *ACM SIGGRAPH*, Vol. 18, No. 3, July 1984, pp. 213-222.
- [20] H. Gouraud, "Continuous Shading of Curved Surfaces", *IEEE Transactions on Computers*, Vol. C-20(6), June 1971, pp. 623-628.

- [21] S. Graham, P. Kessler and K. McKusick, "Gproff", *SIGPLAN Notices* - 82 Symposium on Compiler Construction, Vol. 17, No. 6, June 1982, pp. 120-126.
- [22] D. Halliday and R. Resnick, *Physics Part II*, John Wiley, New York, 1966.
- [23] E. Hecht and A. Zajac, *Optics*, Addison Wesley, Reading, Massachusetts, 1982.
- [24] P. Heckbert and P. Hanrahan, "Beam Tracing Polygonal Objects", *ACM SIGGRAPH*, Vol. 18, No. 3, July 1984, pp. 119-128.
- [25] Mathematical Applications Group, Inc., 3-D Simulated Graphics, *Datamation*, February 1968.
- [26] H. Moravec, "3-D Graphics and the Wave Theory", *ACM SIGGRAPH*, Vol. 15, No. 3, August 1981, pp. 289 - 296.
- [27] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, New York, second edition 1979..
- [28] T. Nishita and E. Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflections", *ACM SIGGRAPH*, Vol. 19, No. 3, July 1985, pp. 23 - 30.
- [29] D. Peachey, "Solid Texturing of Complex Surfaces", *ACM SIGGRAPH*, Vol. 19, No. 3, July 1985, pp. 279 - 286.
- [30] S. Ramo, J. Whinnery and T. Van Duzer, *Fields and Waves in Communication Electronics*, John Wiley, New York, 1965.
- [31] D. Reddy and S. Rubin, Representation of Three-Dimensional Objects, Tech Report(CMU-CS-78-113), Department of Computer Science, Carnegie-Mellon University, 1984.

- [32] W. T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", *ACM Transactions on Graphics*, Vol. 2, No. 2, April 83, pp. 343-349.
- [33] L. G. Roberts, Machine Perception of Three-Dimensional Solids, in *Optical and Electro-Optical Information Processing*, Tipper et al (ed.), MIT Press, May 1963.
- [34] L. G. Roberts, *Homogeneous Matrix Representations and Manipulation of N-Dimensional Constructs*, Lincoln Laboratory, Massachusetts, 1965.
- [35] R. S. Rougelot and R. Shoemaker, G. E. Real Time Display, NASA Rep. NAS 9-3916.
- [36] S. Rubin and T. Whitted, "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", *ACM SIGGRAPH*, Vol. 14, No. 4, July 1984, pp. 110 - 116.
- [37] I. E. Sutherland, in *SKETCHPAD: A Man-Machine Graphical Communication System*, Spartan Books, Baltimore, Md, 1963.
- [38] I. E. Sutherland, R. F. Sproull and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms", *Computing Surveys*, Vol. 6, No. 1, March 1974, pp. 1-55.
- [39] K. E. Torrance and E. M. Sparrow, "Theory for Off-Specular Reflections from Roughened Surfaces", *Journal Optical Society of America*, Vol. 57, No. 9, Sept 1967, pp. 1105-1114.
- [40] J. Ullner, *Parallel Architectures for Computer Graphics*, Ph.D. Thesis, California Institute of Technology, June 1984.

- [41] H. Weghorst, G. Hooper and D. Greenburg, "Improved Computational Methods for Ray Tracing", *ACM Transactions on Graphics*, Vol. 3, No. 1, January 1984, pp. 52-69.
- [42] T. Whitted, "An Improved Illumination Model for Shaded Display", *Comm. ACM*, Vol. 23, No. 6, June 1980, pp. 343-349.